

UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Quantitative analysis of genetic expression responses to dynamic microenvironmental perturbation

Permalink

<https://escholarship.org/uc/item/331662h0>

Author

Pang, Wyming Lee

Publication Date

2007

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Quantitative analysis of genetic expression
responses to dynamic microenvironmental
perturbation**

A dissertation submitted in partial satisfaction of the requirements for the degree
Doctor of Philosophy

in

Bioengineering

by

Wyming Lee Pang

Committee in charge:

Professor Jeff Hasty, Chair
Professor Stuart Brody
Professor David Gough
Professor Alexander Hoffmann
Professor Gabriel Silva

2007

Copyright
Wyming Lee Pang, 2007
All Rights Reserved

The dissertation of Wyming Lee Pang is approved, and it is acceptable in quality and form for publication on microfilm.

Chair

University of California, San Diego

2007

For my wife
Lisa
Thanks for your patience

It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change.

Charles Robert Darwin (1809–1882)
English Naturalist

Table of Contents

Signature Page	iii
Dedication	iv
Epigraph	v
Table of Contents	vi
List of Figures	ix
List of Tables	xii
Acknowledgments	xiii
Curriculum Vitæ	xvi
Abstract	xviii
1 Introduction	1
1.1 Quantitative biology	1
1.2 Model systems	3
1.3 Dynamic environments and living cells	6
1.4 Current techniques for monitoring gene expression	8
1.5 Microfluidic devices	9
1.5.1 Modeling on-chip pressures and flows	10
1.5.2 Device fabrication	16
1.5.3 Driving and controlling fluid flow	18
1.6 Goals	21
2 Platform Development and Methodology	22
2.1 Cells, Constructs, and Culture Conditions	22
2.1.1 Parent and fluorescent variant strains	22
2.1.2 Cellular transformations	27
2.1.3 Steady-state expression characterization	29
2.2 Microfluidic Devices	31
2.2.1 Device fabrication	32
2.2.2 Confining microbial biofilms and microcolonies	36
2.2.3 A quantitative, long-duration imaging platform	44
2.2.4 Controllable dynamic microenvironments	54
2.3 Microscopy	65
2.3.1 System description	65
2.3.2 Vibration isolation	66
2.3.3 Reduction of thermally induced drift in focal plane	67
2.3.4 Autofocus method	67
2.3.5 Image processing and analysis	70
2.4 Acknowledgements	71

3	Metabolic gene regulation in a dynamically changing environment	72
3.1	Introduction	72
3.2	Materials and methods	75
3.2.1	Microfluidic device fabrication	75
3.2.2	Cell preparation and culture	76
3.2.3	Experimental conditions	77
3.2.4	Data analysis	78
3.3	Results and Discussion	79
3.3.1	Dynamic profiling	79
3.3.2	Computational model	80
3.3.3	Computational and experimental analysis of response robustness . .	84
3.4	Conclusions	88
3.5	Acknowledgements	88
4	Summary and Future Directions	89
4.1	Review	89
4.2	Significance	90
4.3	Future directions	90
4.4	Closing	92
A	MOCA: Microfluidic Open Circuit Analyzer	93
A.1	Basic description, requirements, and concepts	93
A.1.1	Requirements	94
A.1.2	Concepts	94
A.2	Simulating a device	97
A.3	Common approximations	100
A.3.1	Channels in parallel	100
A.3.2	Channels in series	101
A.3.3	Valved channels	101
A.4	MOCA models for devices used in this work	102
A.4.1	$T\mu C$	102
A.4.2	$T^2\mu C$	104
A.5	Complete source code for <code>moca.m</code>	109
B	IMSQT: IMage Segmentor, Quantifier, Tracker	117
B.1	Overview	117
B.2	Main Window	119
B.2.1	Experiment Path	120
B.2.2	Correction Path	120
B.2.3	Location ID	120
B.2.4	Indices	120
B.2.5	Regions of Interest	121
B.2.6	Channels	122
B.2.7	Saving and Loading IMSQT sessions	124
B.3	Image segmentation	125
B.3.1	Automated segmentation	125
B.3.2	Manual segmentation using SEGEDIT	128
B.4	Object Quantification	133
B.5	Object Tracking	135

B.5.1	Tracking Validation with TRACKEDIT	138
B.5.2	Trajectory Viewing with TRACKVIEW	143
B.6	Complete source code for IMSQT main window	147
B.7	Complete source code for SEGEDIT	173
B.8	Complete source code for TRACKEDIT	203
B.9	Complete source code for TRACKVIEW	216
B.10	Complete source code for segmentor modules	220
B.11	Complete source code for library functions	234
C	Hardware Platforms	277
C.1	Automated microscopy platform	277
C.1.1	Hardware configuration	277
C.1.2	Generalized light paths	278
C.1.3	Imaging algorithm	279
C.2	Waveform generation platform	281
C.2.1	Hardware configuration	281
C.2.2	Source documentation	282
D	Microfluidic Devices	290
D.1	T μ C: Tesla micro-Chemostat	290
D.1.1	Fabrication	290
D.1.2	Device Schematic and Port Assignments	291
D.2	T ² μ C: Temporal Tesla micro-Chemostat	292
D.2.1	Fabrication	292
D.2.2	Device Schematic and Port Assignments	293
D.2.3	Operation Protocol	294
D.3	Glial Network Stimulator	303
D.3.1	Fabrication	303
D.3.2	Device Schematic and Port Assignments	303
D.4	DynaGrad: Dynamic Chemical Gradient Device	305
D.4.1	Fabrication	305
D.4.2	Device Schematic and Port Assignments	305
E	Computational code for T ² μ C data simulations and analysis	306
E.1	Data analysis	306
E.2	Model simulation code	321
	Bibliography	329

List of Figures

Figure 1.1	Cartoon diagram of the galactose utilization pathway	3
Figure 1.2	Node/Segment schematic of a fluidic “t”-junction	12
Figure 1.3	Node/Segment schematic of a fluidic cross-junction	14
Figure 1.4	Node/Segment schematic of a fluidic h-cross	15
Figure 1.5	Microfluidic device fabrication process	18
Figure 2.1	<i>GAL2</i> expression in <i>S. cerevisiae</i> YPH499, YPH500, and K699.	25
Figure 2.2	Yeast transformation using pKT derived fluorescent fusion protein vectors.	26
Figure 2.3	Steady-state galactose induction for <i>S. cerevisiae</i> YPH499 and K699	30
Figure 2.4	Steady-state glucose repression for <i>S. cerevisiae</i> YPH499 and K699	32
Figure 2.5	Schematic diagram of the “Sticky-Pad” device.	37
Figure 2.6	Patterning of “Sticky-Pads”.	39
Figure 2.7	Cellular growth on “Sticky-Pads”	40
Figure 2.8	Colony comets on a “Sticky-Pad”	41
Figure 2.9	The XLC growth chamber array.	43
Figure 2.10	The $T\mu C$	45
Figure 2.11	$T\mu C$ advection/diffusion analysis schematic	47
Figure 2.12	1D representation of the $T\mu C$ with diffusive and advective transport	47
Figure 2.13	Comparison of model and experimental $T\mu C$ large molecule transport	51
Figure 2.14	Analytical of time evolved 1D concentration profiles under various advective velocities	52

Figure 2.15 Comparison of model and experimental $T\mu C$ small molecule transport	52
Figure 2.16 Nutrient transport in a fully confluent $T\mu C$ microchamber	54
Figure 2.17 Advantages of monolayer imaging	55
Figure 2.18 A dynamically controlled gradient profile device	57
Figure 2.19 The $T^2\mu C$	58
Figure 2.20 Cartoon depiction of laminar interface guidance	60
Figure 2.21 Linearly graded mixing output	60
Figure 2.22 Characterization of on-chip waveform generation	61
Figure 2.23 Nutrient transport in a confluent $T^2\mu C$ growth chamber.	64
Figure 2.24 Loading of the $T^2\mu C$	64
Figure 2.25 Autofocus pattern using dry air chambers and fluorescent illumination.	69
Figure 3.1 Simulated and experimental expression trajectories in response to sinusoidal perturbation at varying frequencies.	80
Figure 3.2 Schematic of coupled galactose and glucose regulatory networks used to derive the computational model.	82
Figure 3.3 Induction/repression dynamic response of <i>S. cerevisiae</i> YPH499 and K699	86
Figure 3.4 Amplitude ratio and phase shift profiles from experimental and simulated data	87
Figure A.1 Node/Segment schematic of a fluidic h-cross	95
Figure A.2 Graphical output of a MOCA simulation of an h-cross microfluidic system. The tooltip is displayed when hovering the mouse pointer over the node label. Similar tooltips are available for the segment labels.	99
Figure A.3 Graphical results for MOCA simulation of the $T\mu C$ microfluidic device	103
Figure A.4 Graphical results for MOCA simulation of the $T^2\mu C$ microfluidic device	105
Figure B.1 The IMSQT main window	119
Figure B.2 Regions of interest definition panel in the IMSQT main window.	121
Figure B.3 Channels definition panel in the IMSQT main window.	122

Figure B.4 Segmentation panel in IMSQT main window	125
Figure B.5 SEGEDIT manual segmentation editor	128
Figure B.6 SEGEDIT subpanels: (a) morphological operations panel, (b) object editing, and (c) morphological filtering.	130
Figure B.7 Object quantification panel in IMSQT main window	133
Figure B.8 Object tracking panel in IMSQT main window	135
Figure B.9 TRACKEDIT object tracking validation viewer	138
Figure B.10A sample of TRACKEDIT object display.	140
Figure B.11The TRACKEDIT display control panel.	140
Figure B.12Default graphical output using TRACKVIEW.	145
Figure B.13Data smoothing using TRACKVIEW.	146
Figure C.1 Imaging optical train and light paths	278
Figure C.2 Waveform generation system configuration	281
Figure D.1 Device schematic for $T\mu C$. Inset displays a magnified view of the growth chamber.	291
Figure D.2 Device schematic for $T^2\mu C$. Insets display magnified views of the growth chamber and on-chip media switch.	293
Figure D.3 Device schematic for glial network stimulation device. Insets display mag- nified views of the growth chamber and on-chip media switch.	303
Figure D.4 Device schematic for dynamic gradient device. Inset displays magnified views of the on-chip media switch.	305

List of Tables

Table 2.1	Yeast parent strains	23
Table 2.2	Excitation and emission spectral maxima of fluorescent proteins	24
Table 2.3	Yeast variants	25
Table B.1	A sample trajectory link table.	142
Table D.1	Master mold feature height specifications. [†] Photoresists are SU-8 unless otherwise specified. [‡] These layers were patterned additively (e.g. no development step between this and the prior layer)	290
Table D.2	Port assignments for T ² μC.	291
Table D.3	Master mold feature height specifications. [†] Photoresists are SU-8 unless otherwise specified. [‡] These layers were patterned additively (e.g. no development step between this and the prior layer)	292
Table D.4	Port assignments for T ² μC.	293
Table D.5	Master mold feature height specifications. [†] Photoresists are SU-8 unless otherwise specified.	303
Table D.6	Port assignments for the glial network stimulation device.	304
Table D.7	Master mold feature height specifications. [†] Photoresists are SU-8 unless otherwise specified. [‡] Bacterial and yeast devices require patterning the gradient outflow channel with adhesion molecules such as polylysine (bacterial) or Conavalin-A (yeast).	305
Table D.8	Port assignments for the dynamic gradient device.	305

Acknowledgments

It's been a long six years and I've been fortunate to have met so many interesting people and had more than my fair share of unique and great experiences. I'm not going to say that the journey was easy, and I'm sure that no one in their right mind would have guaranteed that. In fact, some of the most difficult times I've had to date were during my graduate career. These are moments spent wondering if the sum of who you already are and who you want to be is worth the next step in the plans you've made for yourself. It's cliché I know, but I couldn't have made it here with out the unrelenting support of my colleagues, friends, and family.

First, I owe immense gratitude to my professional colleagues, without whom this work would not have been possible. I thank Dr. Hasty for providing financial support and valuable research advice throughout my doctoral studies, and having the patience to see everything through. In addition, I'm fortunate to have been a part of the integrative, collaborative, and supportive environment provided by the members of the Systems Biodynamics Lab, and likewise, I am indebted to Jesse, Chris, Mike, and Ben who helped me edit and refine this dissertation. Jennifer and Natalie, your skills at the bench are as good as your scientific savvy, and I'll be forever thankful for all your assistance with getting my yeast variants made. Matt and Dmitri, you guys work in numbers, theory, and code like artists in fine oils, and are the best computational model builders I know. Moreover, I would also

like to thank Dr. Schmid-Schoenbein, who has always been both kind and supportive, and the first of many bioengineering faculty to get me truly excited in research and the hunt for scientific answers. Lastly, I would like to thank Dr. Groisman, even though our relationship wasn't the best, he gave me a solid, and reliable foundation in microfluidic design and fabrication, for which I am sincerely grateful.

More than anything I'd like to thank my friends who were always there to rejoice in the successes and help through the unavoidable setbacks be they professional or personal. Jennifer, Chris, Lauren, and Jessica, you were always available at a moments notice for coffee, be it at CUPs for a short break, Mandeville for a walk and a talk, the "triple-S" for a quick snack, or Espresso happy hour at the Grove. There's no need to explain how these moments have helped keep me sane (as well as thoroughly awake). Jared, I'm still counting the times you've come through for me as a close friend from being a groomsman at my wedding, to being that extra set of eyes on a difficult problem and having just the right spare part in a pinch. Pat, Peter, Karen, Jesse, Chris, and Joey, if it weren't for Friday night poker, I'd probably have paid off my car by now, but even if I could, I wouldn't change a thing. Diane, Jared, and Chris, thanks for (trying) to keep me in shape, both on the tennis court and running at the crack of dawn. Ben, when it came to data processing, you always had the time to help as well as the right answer. Mike, you were one of the best sounding boards for all my frustrations, crazy ideas, and personal and professional decisions, by the way, how many burritos have we had?

Above all else, I owe it to my family for your unconditional love and support. To my parents, thank you for supporting my decision to go graduate school to pursue first my masters, and finally my doctorate. Most of all, I'm grateful to have had my wife, Lisa, by

my side since before graduate school. You helped me realize that I'm not just working on my own future, but our shared one, and you've kept me focused and determined every step of the way.

Portions of Chapter 2 appear in part in Cookson S*; Ostroff N*; Pang WL*; Volfson D; Hasty J, "Monitoring dynamics of single-cell gene expression over multiple cell cycles", *Molecular Systems Biology* doi:10.1038/msb4100032, Published online 22 November 2005. The dissertation author is a first author of this publication. Portions of Chapter 2 appear in part in Pang WL; Bennett MR; Ostroff N; Hasty J, "Metabolic gene regulation in a dynamically changing environment", *Nature*, which is in preparation for submission by Feb 2007. The dissertation author is the first author of this publication. The text of Chapter 3 is a reprint of Pang WL; Bennett MR; Ostroff N; Hasty J, "Metabolic gene regulation in a dynamically changing environment", *Nature*, which is in preparation for submission by Feb 2007.

Curriculum Vitæ

Education

- 2007 Doctor of Philosophy in Bioengineering
University of California, San Diego.
- 2002 Master of Science in Bioengineering
University of California, San Diego.
- 2000 Bachelor of Science in Biochemical Engineering
University of California, Davis.

Teaching

- 2001–2003 Teaching Assistant, Department of Bioengineering, University of California, San Diego
Courses: Process Control (2003), Biotechnology Laboratory, (2001–2002, Head TA 2002), Biochemical Engineering (2001)
- 2001 Teaching Assistant, Department of Chemical Engineering, University of California, San Diego
Courses: Separation Processes

Work Experience

- 2001 Assistant Development Engineer, Department of Biomedical Engineering, University of California, Davis
- 2000–2001 Web Developer, Paradux Concepts
- 2000 Laboratory Assistant, Genentech Inc.
- 1999 Research Intern, SUGEN Inc.
- 1998–2000 Research Intern, Process Control Lab, Department of Chemical Engineering, University of California, Davis

1998–1999

Research Intern, Dairy Food Safety Lab, School of Veterinary Medicine, University of California, Davis

Selected Publications

Pang WL*, Bennett MR*, Ostroff N*, and Hasty J (* equal contribution). Metabolic gene regulation in a dynamically changing environment. In preparation.

Grilly C*, Stricker J*, Pang WL, Bennett MR and Hasty J (* equal contribution). A synthetic gene network for tuning protein degradation in *Saccharomyces cerevisiae*. *Molecular Systems Biology*. In review.

Cookson S*, Ostroff N*, Pang WL*, Volfson D and Hasty J (* equal contribution). Monitoring dynamics of single-cell gene expression over multiple cell cycles. *Molecular Systems Biology* doi:10.1038/msb4100032. Published online 22 November 2005.

Zhang Q, Pang WL, Chen H, Cherrington J, Lipson K, Antonian L, Shawver LK. Application of LC/MS/MS in the quantitation of SU101 and SU0020 uptake by 3T3/PDGFr cells. *J. Pharm. Biomed. Anal.* 28:701-9 (2002).

Contributed Talks

Temporal driving of gene regulatory networks using microfluidics: bridging the gap between computational simulations and real-world experimentation with high temporal resolution dynamics. 3rd Annual Conference on Systems and Pathways, Rhodes, Greece (2005)

Posters

Genetic and phenotypic responses to fluctuating microenvironments. Fifth Annual International Symposium: Systems Biology and Medicine. Institute for Systems Biology, Seattle, Washington, USA. (2006)

Systems dynamics on a chip. 3rd International Conference on Pathways, Networks, and Systems: Theory and Experiments. October 2–7. Rhodes, Greece. (2005)

ABSTRACT OF THE DISSERTATION

Quantitative analysis of genetic expression responses to dynamic
microenvironmental perturbation

by

Wyming Lee Pang

Doctor of Philosophy in Bioengineering

University of California, San Diego, 2007

Professor Jeff Hasty, Chair

Dynamic environments are commonplace in the natural world, from fluctuations in nutrient sources that control metabolic rates, to radiative cycling that drives circadian rhythms, to mechanical stresses that reform vasculature. So, intuitively one would assume that the regulatory systems that control cellular behavior are acutely adapted to respond to such variable conditions in a robust and appropriate fashion. Yet, despite their potential to provide increased quantitative detail and insight to the natural behavior of cells, highly dynamic perturbations are rarely utilized in the analysis of cellular gene expression and regulation. Part of this stems from the lack of technologies that enable such studies. However, recent advances in microfluidic devices designed to address biologically relevant questions promise to fill this void. Moreover, recently discovered knowledge that the galactose metabolism in *S. cerevisiae*, and possibly similar pathways, are in fact rudimentary

memory systems, strengthens the need for the ability to examine gene regulation under complex and dynamic stimulation.

In this project, microfluidic technology was developed specifically for isolating, observing, and dynamically probing colonies of model host microbes. The devices created not only sustain cells under ideal growth conditions, but do so in a way that allows for long duration acquisition of highly resolved time evolved gene expression within single cells. Furthermore, these imaging capabilities were coupled to a novel microfluidic system that was able to produce precise and continuous concentration waveforms. The microfluidic platform was then utilized to explore the dynamic response profile of the galactose utilization pathway in *S. cerevisiae* under fluctuating nutrient conditions. Using experimental data, this study revealed that the pathway kinetics lead to low-pass information filtration. Further experimental investigation coupled with computational model simulations uncovered coupling to glucose metabolism that provides a globally robust response, despite galactose utilization impairment. These results emphasize both the utility of microfluidic device platforms in quantitative biological studies, and the importance of studies conducted in more natural environments for gaining a more detailed understanding of how gene systems result in complex behavior.

1

Introduction

1.1 Quantitative biology

As an undergraduate in biochemical engineering at UC Davis, I took the upper division biochemistry and cell biology course series offered by the Biology department to fulfill technical elective requirements. One of these courses, Biology 104, covered cellular pathways and their role in cellular function. In almost every lecture of this class we would learn about a new cellular pathway, and with equal frequency, the instructor would begin the lecture with "...this pathway is, well, sort of complicated ...". In hindsight, this view of biology was understandable. It was the latter half of 1998, and at the time, only a few microbial genomes (*E. coli* K12-MG1655 and *S. cerevisiae* S288C), along with a few viruses, had published sequences[14, 20, 63]. In the same chord, assays using green fluorescent protein (GFP) were relatively new, and gene expression microarrays had only recently been commercialized a couple years prior. Much of the information in my course text had been gathered by nearly half a century of meticulous experimental work, but the components list needed for a systems view of biology was far from complete. It seemed that

every new discovery in the biological fields added more nuance and intricacy to an already complex foundation, pushing a fundamental understanding of “the way things worked” further into the future and out of reach.

Since that time, there have been major advances in biotechnology such as the proliferation of microarray technologies and the completion of genome sequences for all major model organisms. Even so, the prediction of cellular behavior remains a significant challenge, despite a reasonably extensive knowledge of the underlying gene networks involved. Our view of cellular signaling and regulatory pathways is no less complex than it was before. The construction and analysis of synthetic systems has provided unique insights regarding the function of their native counterparts[41, 30, 37]. Yet, even the most carefully designed synthetic network is burdened with inherent stochastic fluctuations, both from external sources and within the network itself, making accurate, deterministic prediction difficult.

Studies of simple, single gene systems, such as autoregulatory feedback loops, demonstrate the power of stochastic fluctuations in determining the system state[9, 8, 49]. Results from these and similar studies have led several to focus on both characterizing and isolating the sources of these perturbations[79, 31, 13, 109, 99, 82] as well as harnessing them for the development of robustly stable genetic circuits[117, 1, 88, 38, 75]. By successively adding complexity to basic systems, complex networks can be probed, characterized, and optimized in a controllable and iterative fashion[38]. Such an approach lays the foundation for the future development of “designer cells” in which desired functionality is preprogrammed into their genomes. At the root of this goal is the requirement of a fundamental set of predictive laws, much like those found in chemistry, physics, and engineering. The achievement of these laws defines quantitative biology.

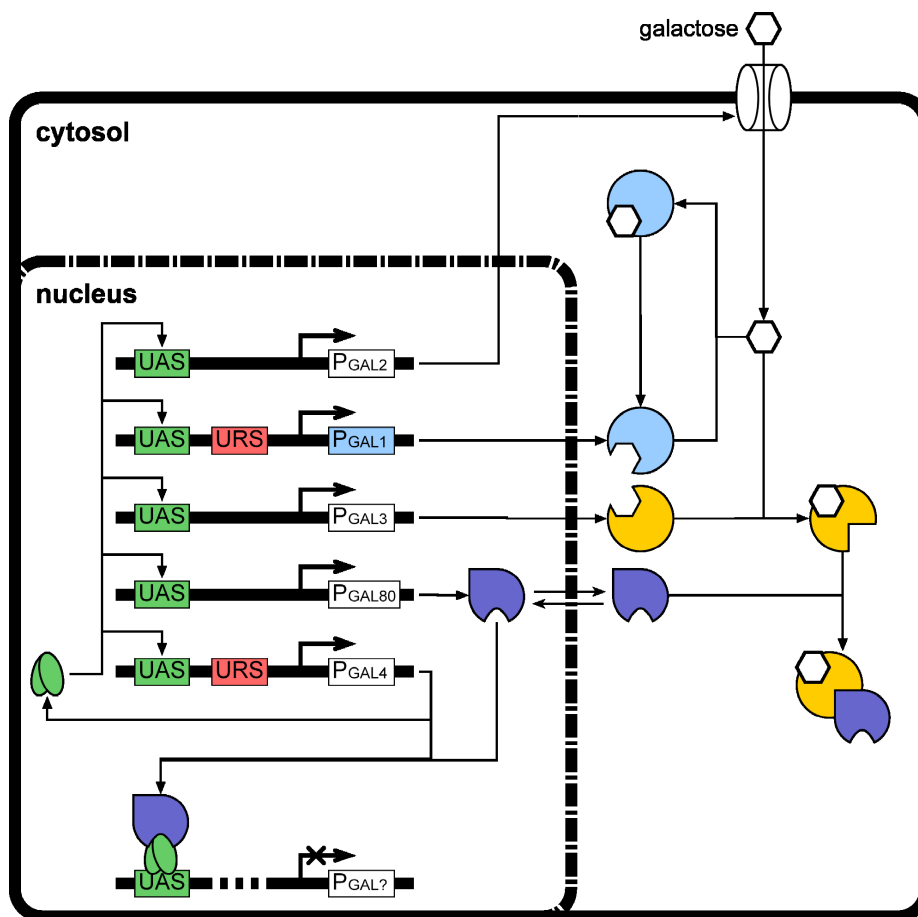


Figure 1.1: Cartoon diagram of the galactose utilization pathway, abridged to the sensory, regulatory, and committal step towards galactose metabolism.

1.2 Model systems

In the same way an artist draws inspiration from what he or she has experienced, continued advances in quantitative and synthetic biology rely on the study of model genetic systems. Moreover, the discrete analysis of gene regulatory networks often reveals novel phenomena in “well understood” gene systems. For example, galactose utilization (Figure 1.1) is one of the best understood inducible metabolic pathways in *S. cerevisiae*[120]. In the same note, it is also an elegantly designed genetic switch[84] and for these reasons, it is an ideal target for many studies in gene regulatory dynamics[47, 109].

When a yeast cell encounters galactose in its environment (and in the absence of catabolic repressors), the galactose is transported from the external environment into the cell via Gal2p, a hexose transporter. Gal2p is known to have two modes of operation, active under low concentrations of galactose, and passive or diffusive under high concentrations of galactose[87]. Once inside the cell, galactose binds to and activates Gal3p, a complementary protein to the galactokinase Gal1p without galactokinase activity, allowing it to bind to cytosolic Gal80p, a transcriptional repressor of *GAL* genes which freely shuttles between the cell nucleus and the cytosol. When in the nucleus, Gal80p is typically bound the Gal4p, the pathway's transcriptional activator, masking an RNA polymerase recruitment site found on non-DNA binding domain of Gal4p. By binding to Gal3p, Gal80p is sequestered to the cytosol, allowing for constitutively bound Gal4p to recruit RNA polymerase. This induces production of Gal1p, the committal step to galactose metabolism, as well as increased production of Gal2p, Gal3p, and Gal80p.

The core switching mechanism within the galactose pathway is the toggling of Gal80p localization. While this binding affinity may in fact be quite weak, the state of the system is reinforced by the positive feedback loop created by galactose, Gal4p, and Gal3p[11]. The induced state is strengthened further by a parallel positive feedback loop created by Gal4p and Gal2p[43]. The solitary negative feedback loop formed by Gal80p serves to destabilize the main positive loop formed by Gal3p, allowing the system to return to a non-induced state when removed from a galactose environment. Thus, the structure of the galactose “switch” appears to create a rudimentary memory system for the yeast cell. Work recently published by Acar and Van Oudenaarden[1] explores this memory effect. In their findings, the positive feedback redundancy makes the pathway initially

incredibly sensitive to galactose. However, once the cell has been fully induced, small changes in the galactose concentration do not affect the system state. Thus the system in its native form appears robust to small scale environmental changes. More interestingly, the amount of persistence can be tuned by the intracellular concentration of Gal80p. Reduced concentrations of Gal80p lead to increasingly persistent induced states to the point where a monostable “always on” state is reached.

Natively the galactose pathway is tightly coupled to glucose repression, an equally well studied system whose signalling pathway is also a multi-feedback information switch[54]. The glucose signalling response is controlled by several transcriptional repressors, Rgt1p, Std1p/Mth1p, Mig1p, and Mig2p. Signaling begins with detection by glucose specific sensors Snf3p and Rgt2p. Although structurally similar to hexose transporters, they do not import glucose into the cell. Instead, they relay information of the glucose binding event to a membrane bound kinase, Yck1p, triggering it to phosphorylate transcriptional co-regulators Mth1p and Std1p. Once phosphorylated, Mth1p and Std1p are targeted by the ubiquitin-ligase SCF^{Grr1} for degradation. In the absence of glucose, Mth1p and Std1p are recruited by Rgt1p. This regulatory complex inhibits the expression of glucose specific hexose transporters (Hxt1-4) as well as Snf3p and Mig2p.

Unique to the glucose response is its ability to adapt to multiple concentrations of external inducer. When freely expressed in a high glucose environment (2–10% w/v), Mig2p represses the expression of Hxt2p and Hxt4p, which are high affinity glucose transporters. Mig2p also represses expression of Snf3p and Mth1p, and has weak repressive control over other carbon utilization genes (SUC, MAL, GAL). Yet, under low to moderate levels of glucose (>0.1%), basal activity of Rgt1p inhibits Mig2p expression, allowing all four trans-

porters to be expressed. Once glucose enters the cell, it mediates the dephosphorylation of cytosolic Mig1p allowing it to shuttle into the nucleus where it inhibits expression of other carbon utilization pathways via the Ssn6-Tup1 repressor complex[120, 26].

The control loop formed by glucose, Std1p/Mth1p+Rgt1p, and the Hxt's are, for the most part, a classic negative feedback scheme, while the loop formed by glucose, Std1p/Mth1p, and Std1p/Mth1p degradation is an interestingly formulated positive feedback motif. The main switching mechanism for the glucose response and the resultant catabolite repression is the rapid change in phosphorylation state of Mig1p. This is tightly synchronized with upregulated expression and degradation of Std1p and Mth1p which results in a sequestration by degradation mechanism thought to allow for rapid silencing of the signaling network when glucose has been depleted[55, 80, 81].

1.3 Dynamic environments and living cells

The intricate nature in which the glucose and galactose pathways in *S. cerevisiae* are coupled could not have arisen in an organism conditioned to a steady environment. Instead, it is known that biological systems in their natural environments are exposed to a myriad of complex stimuli, such as changes in nutrients, growth factors, radiative cycling, and mechanical forces[60]. However, under laboratory conditions, complex external forces are either minimized or eliminated altogether, bringing one to question whether the behavior of biological systems in the lab are representative of their true nature.

Up to this point, the experimental study of gene regulatory networks has primarily relied on relatively simple means of system perturbation, such a single step or pulse input within a timecourse[15, 90]. Yet, recent theoretical work has shown that more quan-

titative analysis of a system's dynamics is possible with perturbations that are sinusoidal in nature[69]. For instance, in order to characterize dynamic processes, engineers turn to a frequency response analysis. A process subjected to an oscillatory input perturbation will subsequently produce an oscillatory response. This response will vary in amplitude and phase-shift relative to the input in a way that is characteristic of the dynamics of the underlying process. By successively probing the process with multiple input frequencies, one can generate a dynamic profile, or frequency response, consisting of a trend of output amplitudes and phase-shifts with respect to input frequency. This data can then be used to optimize process gain and temporal sensitivity.

From a biological perspective, these same measures can be used to characterize a gene expression response and determine how it is affected by other factors such as gene mutations, deletions, and environmental variation. In addition, frequency responses can help to highlight phenomena, such as systems resonances, which are unobservable by simple step or pulse experiments, or even determine previously uncharacterized network topology[69].

In the context of the model systems described above, there has been much research exploring the dynamics of the galactose pathway[15, 108, 66, 94, 47, 90]. The Acar and Van Oudenaarden study, along with other computational work exploring the fate of cellular populations growing under stochastic environments[98, 62, 61] highlights the fact that dynamic perturbation of gene systems is required for advancing quantitative biological research.

1.4 Current techniques for monitoring gene expression

There are currently many windows into the secret workings of the cell, extending from DNA, to RNA, and finally to proteins and their enzymic functions. The predominant trend has been the use of fluorescent reporters, such as the green fluorescent protein and its growing set of variants[17, 22, 93, 103], coupled to high throughput technologies such as flow cytometry and gene microarrays. These and similar techniques offer the ability to probe many cells and many expression states with incredible precision and in a time efficient manner. Due to these attributes, they have become the workhorses of systems biology research[57, 106, 111, 114, 33, 44].

However, despite their broad spectrum applications, and refined precision, these techniques are merely snapshots of cellular expression states, and the limitations of these technologies are exposed when more information regarding gene expression dynamics is desired. For instance, analysis of an inducible expression system on a flow cytometer is straightforward. The populations of cells are stimulated with varying amounts of inducer and fluorescence readouts are generated. A similar mode of analysis can be done on a gene or protein microarray if a complete cellular profile is required. In the case of a system that exhibits bistability or inherent oscillations, the fluorescence readout would be either bimodal, or have an extremely large variance. While informative of the fact that the system being analyzed is different from one that has only one stable expression state, flow cytometry lacks fine grained detail regarding how the two observed states exist and how they might temporally behave. Finally, consider the case of gene expression under dynamic perturbation using smooth waveforms. While conceptually possible, such an experiment of this nature is at the limits of current technological capabilities. The flow cytometer typ-

ically takes 1–5 minutes to extract data from individual samples. A statistically relevant time point would therefore require a minimum of 5–15 minutes to retrieve. At this limited sampling rate, temporal characteristics occurring at timescales faster than this, such as noise properties or feedback ringing, are effectively lost. Even mapping step and pulse responses with sufficient temporal resolution poses a difficult technical challenge[15, 90].

1.5 Microfluidic devices

The emergence of microfluidic devices as quantitative analysis platforms has greatly advanced research capabilities and productivity via miniaturization, technological integration, parallelized scalability, and cost effective sample efficiency[86, 101]. In their simplest form, microfluidic devices are merely microscale channels with inlets and outlets for interfacing with the macroscopic world. Additional functionality is built into devices by adding additional microchannels and/or microchambers for either parallelizing on-chip processes, storing and processing reaction species, or controlling fluid flow[105].

Some of the most impressive developments have only recently been published and include a set of microfluidic platforms for crystallizing proteins and determining their structure[40], a fully integrated device for DNA extraction and purification[46], and several microfluidic platforms for observing and quantifying microbial growth and gene expression[4, 36, 21].

Increasingly, microfluidic platforms are being used to investigate fundamental biological questions such as cellular migration[48, 91], responses to shear stress and other mechanical stimuli[71, 52, 53], and the specifics of stem cell differentiation[102, 19]. Because they can easily isolate small reaction volumes and manipulate single cells, microflu-

idic devices are an ideal platform for studying phenomena such as the dynamics of protein expression[16], long duration gene expression[6, 21, 100], and population dynamics[4, 5, 83], since they can easily isolate small reaction volumes and manipulate single cells. More importantly, they provide investigators the ability to precisely and dynamically manipulate the local environment[68, 18, 50, 65] without sacrificing temporal resolution in reporter read-outs. Thus, current developments in microfluidic devices represent a convergence of fluid dynamics, physics, and microscale integration with life science research[10, 67, 21, 59, 119].

1.5.1 Modeling on-chip pressures and flows

Microfluidic devices can be characterized using fluid flow simulations with relative ease. This is useful in both the design and evaluation of device performance, allowing for efficient development cycles, saving costly materials and time spent on fabrication work. Starting with the Navier-Stokes equations for incompressible fluid flow,

$$\nabla \cdot \mathbf{v} = 0 \tag{1.1}$$

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v} \tag{1.2}$$

If one assumes the following dimensionless parameters,

$$U = \left(\frac{v}{u_0} \right) \tag{1.3}$$

$$\Theta = \left(\frac{tu_0}{L} \right) \tag{1.4}$$

$$\nabla = L \nabla \tag{1.5}$$

$$\mathcal{P} = \left(\frac{p - p_0}{\rho u_0^2} \right) + \frac{\phi}{u_0^2} \tag{1.6}$$

$$N_{\text{Re}} = \frac{\rho u_0 L}{\mu} \tag{1.7}$$

equations 1.1 and 1.2 become,

$$\nabla \cdot \mathbf{U} = 0 \quad (1.8)$$

$$\frac{\partial \mathbf{U}}{\partial \Theta} + \mathbf{U} \cdot \nabla \mathbf{U} = -\nabla \mathcal{P} + \frac{1}{N_{\text{Re}}} \nabla^2 \mathbf{U} \quad (1.9)$$

Under steady state conditions, e.g. when there is no local acceleration of fluid particles, equation 1.9 is further simplified to,

$$\mathbf{U} \cdot \nabla \mathbf{U} = -\nabla \mathcal{P} + \frac{1}{N_{\text{Re}}} \nabla^2 \mathbf{U} \quad (1.10)$$

It is important to now note that due to the length scales associated with microfluidic flows (on the order of 10–100 μm), $N_{\text{Re}} \ll 1$. Under these conditions, it is clear from Eqn. 1.10 that the viscous term $\frac{1}{N_{\text{Re}}} \nabla^2 \mathbf{U}$ dominates over the internal term $\mathbf{U} \cdot \nabla \mathbf{U}$. Thus it is safe to assume that Eqn. 1.10 can be simplified one last time to,

$$\nabla \mathcal{P} = \frac{1}{N_{\text{Re}}} \nabla^2 \mathbf{U} \quad (1.11)$$

By solving Eqn. 1.11 for flow through a cylindrical pipe of radius r and length L , one arrives at following solution,

$$\Delta P = Q \cdot \left(\frac{8\mu L}{\pi r^4} \right) \quad (1.12)$$

which is similar to Ohm's law for analog electrical circuits,

$$V = I \cdot R \quad (1.13)$$

A similar analysis done on a channel with a rectangular cross-section[10] yields,

$$\Delta P = Q \cdot \left(\frac{12\mu L}{wd^3} \right) \cdot \alpha \quad (1.14)$$

where,

$$\alpha = \left[1 - a \left(\frac{192}{\pi^5} \sum_{n=1,3,5}^{\infty} \frac{1}{n^5} \tanh \left(\frac{n\pi}{2a} \right) \right) \right]^{-1} \quad (1.15)$$

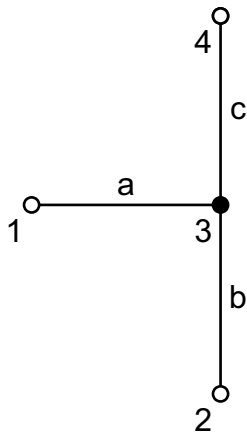


Figure 1.2: Node/Segment schematic of a fluidic “t”-junction

and,

$$a = \frac{d}{w} \quad (1.16)$$

Note for small values of a , i.e. $w \gg d$, α tends toward unity and the fluidic resistance through such a rectangular channel is simply,

$$R_f = \frac{12\mu L}{wd^3} \quad (1.17)$$

While Eqns. 1.14 to 1.17 are adequate to describe flow through a single microfluidic channel, the majority of microfluidic devices are comprised of complex networks of inter-connecting channels. First, consider the simple flow system, a fluidic t-junction, depicted in figure 1.2. Because the fluid flow through the system follows an analog of Ohm’s law, analysis of the system is similar to that of an analog electrical circuit. The system contains three external nodes (open circles) and one internal node (black circle). It is assumed that all external nodes are specified, as would be the case for an actual microfluidic device where the external nodes (fluidic access ports), are connected to external fluid and pressure supplies. In order to fully characterize the flow pattern within the device one must determine

the direction and magnitude of flow in each of the three segments, as well as the pressure of the internal node (node 3). This can be done using an analysis similar to Kirchoff's current law where the electron flux through all circuit nodes must sum to zero. In the case of fluidic systems, fluid flow, represented by Q , replaces electron flux. Thus,

$$\sum_i^n Q_i = 0 \quad (1.18)$$

In this scenario, a positive value for Q_i indicates fluid flowing into the node and a negative value indicates fluid exiting the node. For the system depicted in Fig. 1.2, Eqn. 1.18 becomes,

$$Q_a + Q_b + Q_c = 0 \quad (1.19)$$

Substituting Eqn. 1.14 into Eqn. 1.19 gives,

$$G_a(P_3 - P_1) + G_b(P_3 - P_2) + G_c(P_3 - P_4) = 0 \quad (1.20)$$

where,

$$G_i = \frac{1}{R_i} \quad (1.21)$$

and represents the fluidic conductance of segment i . Solving for the internal node, P_3 gives,

$$G_{abc}P_3 = G_aP_1 + G_bP_2 + G_cP_4 \quad (1.22)$$

where,

$$G_{abc} = G_a + G_b + G_c \quad (1.23)$$

Now consider the system depicted in Fig. 1.3, which is the same system shown in Fig. 1.2 with the addition of an additional external node and connecting segment. Performing the same analysis given by Eqns. 1.19–1.22 yields,

$$G_{abcd}P_3 = G_aP_1 + G_bP_2 + G_cP_4 + G_dP_5 \quad (1.24)$$

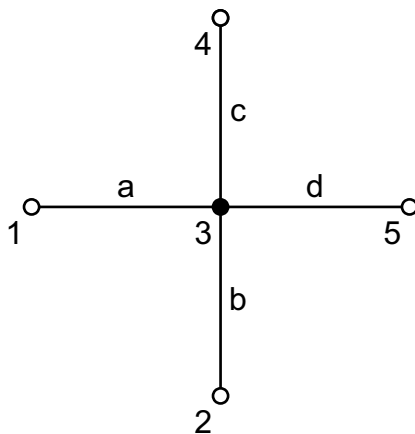


Figure 1.3: Node/Segment schematic of a fluidic cross-junction

By comparing the solutions presented in Eqns. 1.24 and 1.22, several characteristics of the analysis begin to emerge. First, the prefactor for the pressure of any internal node is merely the sum of the conductances of all attached segments. Second, the RHS of the equation that defines an internal node is simply the sum of the neighboring node pressures multiplied by their respective segment conductances.

For more complex fluidic systems, e.g. those having multiple internal node points, a system of equations must be solved to fully define on-chip flow. For example, take the h-cross system shown in Fig. 1.4.

Performing a microfluidic open circuit analysis described above gives the following system of equations,

$$\begin{pmatrix} G_{abc} & -G_c \\ -G_c & G_{cde} \end{pmatrix} \begin{pmatrix} P_2 \\ P_5 \end{pmatrix} = \begin{pmatrix} G_a & G_b & 0 & 0 \\ 0 & 0 & G_d & G_e \end{pmatrix} \begin{pmatrix} P_1 \\ P_3 \\ P_4 \\ P_6 \end{pmatrix} \quad (1.25)$$

pre-multiplying both sides of Eqn. 1.25 by the inverse of the LHS constant matrix gives the

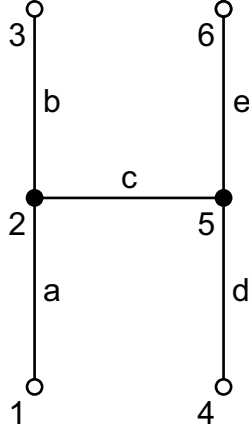


Figure 1.4: Node/Segment schematic of a fluidic h-cross

solution for the internal node pressures,

$$\begin{pmatrix} P_2 \\ P_5 \end{pmatrix} = \begin{pmatrix} G_{abc} & -G_c \\ -G_c & G_{cde} \end{pmatrix}^{-1} \begin{pmatrix} G_a & G_b & 0 & 0 \\ 0 & 0 & G_d & G_e \end{pmatrix} \begin{pmatrix} P_1 \\ P_3 \\ P_4 \\ P_6 \end{pmatrix} \quad (1.26)$$

Now that all the node pressures are known, determining the segment flows simply requires applying Eqn. 1.14 again. First, however, it is necessary to generalize it for a system of equations,

$$Q_i = G_{ij}C_{jk}P_k \quad (1.27)$$

where G_{ij} is a matrix with the segment conductance values as elements and C_{jk} is a constant matrix that specifies segment connections between the nodes P_k . In the case of

the h-cross system described above, Eqn. 1.27 becomes,

$$\begin{pmatrix} Q_a \\ Q_b \\ Q_c \\ Q_d \\ Q_e \end{pmatrix} = \begin{pmatrix} Ga & 0 & 0 & 0 & 0 \\ 0 & Gb & 0 & 0 & 0 \\ 0 & 0 & Gc & 0 & 0 \\ 0 & 0 & 0 & Gd & 0 \\ 0 & 0 & 0 & 0 & Ge \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{pmatrix} \quad (1.28)$$

The process for determining on-chip pressures and flowrates is easily automated, making the simulation of much more complex fluidic systems an efficient process. Furthermore, the fluidic designer need only specify external pressures, segment geometry, and segment connectivity, making it possible to achieve a feasible design without repeated device fabrication and testing. To facilitate microfluidic development, I developed a small utility for simulating microfluidic device designs which is described in more detail in Appendix A. It is important to note that for the devices described in this work, such “macroscopic” flow modeling of device designs was sufficient to achieve reliable real-world estimates of device operational characteristics. In addition, the geometry of fluidic components was kept as simple as possible, e.g. straight walled channels of rectangular cross-section. Thus, more complicated flow modeling techniques, such as computational fluid dynamics methods, were not required.

1.5.2 Device fabrication

Many of the early microfluidic devices were constructed by selective etching of glass and silicon substrates. While state of the art for the time, this process was both dangerous and time consuming, and limited the number and geometry of topographic features on

the device. Moreover, these early glass devices were often thermally annealed to another glass substrate at $\geq 400^\circ\text{F}$ to seal flow channels, making it difficult to pre-pattern biological moieties to channel surfaces.

More recently, many have turned to rapid prototyping techniques that utilize replica molding of polydimethylsiloxane (PDMS), a clear silicone elastomer, against a photolithographically manufactured master mold[113, 112, 116]. This method borrows well established UV photolithography of positive or negative tone resist films on silicon substrates, allowing device designers to utilize a broad range of channel geometries and topologies. Furthermore, PDMS is easily bonded to glass or silicon substrates via simple chemical modification steps that can be carried out at room temperature, making it a more compatible material for biological surface modification.

The fabrication of PDMS microfluidic devices from a concept drawing can take as little as one day to complete under ideal conditions. More complex devices, such as those incorporating pressure actuated valves, micropumps, and other three dimensional structures[101, 105, 78, 2] can take anywhere from 2–3 days to manufacture. In addition, other groups have started to move beyond the use of PDMS as a material for rapid prototyping. New materials include mylar sheets “cut” with a low power carbon-dioxide laser and assembled in a laminating process to form complete devices, precision milled and injection molded polymethylmethacrylate, and chemically etched polyimide. However, PDMS remains the most versatile because of its transparency, mechanical properties, and biocompatibility.

This work utilized the well documented technique of PDMS rapid prototyping via replica molding. Device fabrication by this method occurred in three distinct phases:

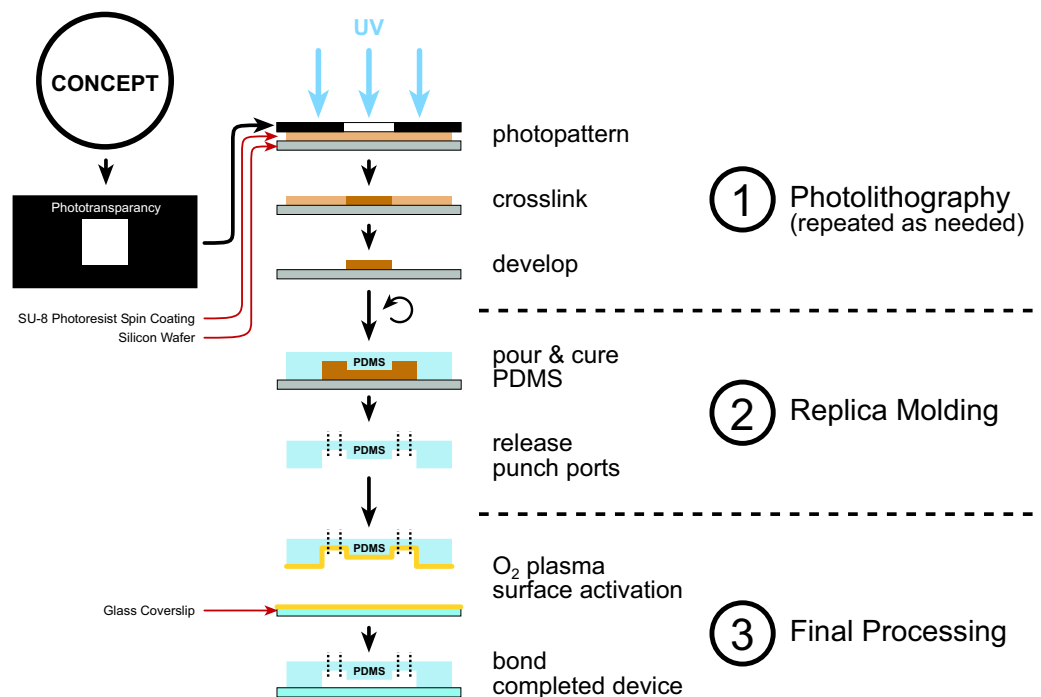


Figure 1.5: Microfluidic device fabrication process

photolithography, replica molding, and bonding/finishing (Fig. 1.5). Prior to fabrication, device designs were drawn in standard vector graphics utilities (e.g. AutoCAD, Autodesk Inc.) and printed as high resolution phototransparencies for use as photomasks in the photolithographic process. The overall fabrication process typically took 3–5 days to complete given the specific equipment and laboratory conditions available and depending on the complexity of the device being fabricated. More specific details of this procedure are given in Chapter 2, Section 2.2.

1.5.3 Driving and controlling fluid flow

Methods for driving fluid flow on microfluidic devices vary greatly, requiring careful consideration of the environment the device will be used in and the data it is intended

to generate. Because, fluid transfer volumes on microdevices range from picoliters to microliters per second they are much more susceptible to mechanical noise than macroscopic flows on the order of milliliters per second or higher. The most commonly cited method for driving on-chip flows is the use of mechanically driven pumps which apply a constant linear force on the plunger of a fluid filled syringe. Output flow is controlled by the speed at which a central driving screw turns, and pumps now exist that are capable of controlling flows on the order of nanoliters per hour. However, such precise syringe pumps are quite costly, ranging from \$5–10K depending on available features, such as the ability to draw and dispense simultaneously. Moreover, a typical syringe pump usually accommodates only 2 independently driven flows. Lower cost pumps exhibit a fair amount of mechanical noise at the microscale, observed as persistent pulsatility in the flow, which is generally an undesired characteristic.

More recent developments utilize the flux of ions in solution to effectively drag the bulk fluid down an electric field[25]. This electrically driven flow, or EOF for Electro-Osmotic Flow, requires the placement of electrodes at reservoir entry and exit ports which are then electrified with 1–10KV to generate ion flux. While ideal for driving flow for chemical reaction chips, it is unknown how the constant application of high potential electric fields affects the viability of living cells. In addition, EOF device designers must take in to careful consideration joule heating effects[32], to ensure that biological species are not cooked *in-situ*. Lastly, since EO flows are dragged by moving ions near the channel walls, their flow profile is plug-shaped and non-laminar[29, 85].

By adapting integrated microvalve technology, it is possible to incorporate microscale peristaltic pumps directly into microdevice. This not only provides exquisite con-

trol over on-chip flows, but is subject to parallelization, and reduces external flow control hardware to a simple regulated air supply and a bank of computer controlled solenoid air valves. In most cases, the pulsatile nature of this micropump is negligible, and many groundbreaking microfluidic technologies have incorporated this form of flow driving. However, there are several notable drawbacks. While, the pump discharges fluid smoothly at high operating flowrates, at lower rates, the pulsatility of the pumping mechanism increases dramatically, limiting the usable output range of the micropump on devices that require smooth, continuous flow.

More exotic methods of driving microfluidic flow utilize capillary action[121] and internal droplet pressure[110]. These methods require no external driving and control mechanisms, and instead rely on the physical properties of the fluid being transported. This makes these methods more suitable for field use or point-of-care diagnostic microfluidic devices, where it is unfeasible to transport an external driving system. In the same respect, capillary action and internal droplet pressure are not completely suitable for use in long duration and repeated use analysis platforms.

The absolute simplest method of driving on-chip flow is the use of hydrostatic pressure heads on connected fluid reservoirs[72, 28, 76]. This method has no mechanical pumping action, eliminating flow pulsatility. Instead, flow is driven either by a regulated air or vacuum supply, or by simple gravity feeding. Under gravity feed, the net pressure driving flow is the differential applied at inlet and outlet reservoirs, and can be precisely controlled to within 1/100th of a psi using “gravity towers” that suspend fluid reservoirs at calibrated heights. While these towers can be quite bulky, they offer the most flexibility in controlling fluid flowrates, which is invaluable in the design and testing of novel microfluidic

devices.

1.6 Goals

The development of mathematical principles, theory, and predictive models for biological systems requires the ability to study such systems as they respond to highly dynamic environments. The proper tools will provide experimental data that quantitatively complements computational studies, possibly leading to the discovery of new phenomena and behavioral structures in gene networks. In this work I describe how I set forth in realizing both the experimental platform and the associated scientific benefits.

I began by building a microfluidic platform for monitoring gene expression over long durations which also provided precise, yet dynamic control over the microenvironment. There were many technical challenges, both engineering and biological, that were overcome and are discussed in detail in Chapter 2 and in supplementals provided in the appendices. The platform was then applied to the study of the galactose utilization system in Chapter 3. This study differed from previously published work in that it used a truly dynamic mode of perturbation — pure sinusoids of glucose concentration rather than solitary step or pulse changes. Here I experimentally and computationally examined the differences between the frequency response profiles of two phenotypically different strains of yeast. This not only revealed that the galactose utilization system is a robust information filter, but is also more tightly coupled to glucose repression than previously believed. Conclusions from the platform development and dynamic response study are presented in Chapter 4. Lastly, future outlook and insights on applications of my platform to other systems is provided along with closing remarks.

2

Platform Development and Methodology

2.1 Cells, Constructs, and Culture Conditions

2.1.1 Parent and fluorescent variant strains

S. cerevisiae was chosen as the model organism because of the abundance of knowledge regarding its galactose utilization pathway[120] and relatively straight forward methods for modifying its genome[51]. More specifically, *S. cerevisiae* K699, YPH499 and YPH500 (see Table 2.1 for genotypes) were used because they contained deletions of several metabolic pathways allowing for the use of auxotrophic selection markers in genetic transformations[95]. These strains were commonly utilized in the yeast literature and are phenotypically different in several ways. Because of their descendance from the S288C strain, the two YPH strains, 499 and 500 do not utilize galactose anaerobically[77]. Moreover, and possibly related, YPH cells were observed to grow approximately 50 percent slower

Table 2.1: Yeast parent strains used for genetic transformations.

Designation	Mating Type	Genotype
YPH499	a	<i>ura3-52 lys2-801_{amber} ade2-101_{ochre} trp1-Δ63</i> <i>his1-Δ200 leu2-Δ1</i>
YPH500	α	<i>ura3-52 lys2-801_{amber} ade2-101_{ochre} trp1-Δ63</i> <i>his1-Δ200 leu2-Δ1</i>
K699	a	<i>ade2-1 trp1-1 leu2-3 leu2-112 his3-11 his3-15</i> <i>ura3 can1-100</i>

than K699 cells in equivalent, non-glucose media. The defining phenotypic manifestation of this difference in galactose utilization was an experimentally observed 10-fold increase in the amount of galactose required to induce the YPH cells to a level comparable to that seen in the K699 strain[70]. While many attribute this to a deficiency in *GAL2*, the gene coding for the galactose specific hexose transporter, sequencing work performed by our lab show no mutations to this gene that could cause altered protein function[56]. Other studies exploring this phenotypic difference point to a recessive mutation in *IMP1* a gene thought to be allelic to *GAL2*[104]. Lastly, YPH cells contain a deletion in the adenine biosynthesis pathway. Without this supplemented in the media, intracellular adenine biosynthesis bottle-necks at a flavin precursor which accumulates causing a reddish pigmentation within the cell[7, 35]. While this does not cause any noticeable deleterious affects on growth, it does create a significantly large intracellular background when imaged using 558 nm light, the excitation band used in DsRed and red fluorescent tracer illumination (about 10-fold higher than non-cellular image background).

To monitor the response of the galactose pathway, several yeast variants expressing several combinations of yEGFP, yEVENUS (YFP), and yCerulean (CFP) (Table 2.3) were created using modified pRS shuttle vectors[109, 95], and pKT integrating vectors[93],

Table 2.2: Excitation and emission spectral maxima of fluorescent proteins. *SR101 is sulforhodamine 101, a low molecular weight, hydrophilic, red fluorescent dye that is used to track on-chip concentrations of inducing or repressing chemical species, or in microfluidic flow validation procedures.

Protein	Excitation (nm)	Emission (nm)
yEGFP	488	507
yEVenus	515	530
yCerulean	434	477
DsRed	558	583
SR101*	586	605

pKT0090 and pKT0101a (Fig. 2.2), respectively. These vectors contained a flexible linker sequence fused to fluorescent protein domain upstream of a histidine (HIS5) auxotrophic marker gene. The two color variant, WLPY012, was created by splicing a *URA3* coding region in place of the histidine marker on the pKT0090 vector and performing sequential transformation and selection steps.

Several pilot studies characterizing cellular growth in batch culture and in microfluidic devices were carried out using yEGFP and yEVenus fusions to the Gal1 protein, however, later studies used yCerulean fusions. This substitution was made because of its increased spectral separation from sulforhodamine 101 (a.k.a. SR101, Sigma), a red fluorescent dye used to track inducer concentrations (Table 2.2). No deleterious effects from this change were observed.

YPH Gal2p-yEVenus variants (WLPY008 and WPLY009) were created to verify Gal2p deficiency in the YPH lineage. When microscopically imaged in media containing 2% (w/v) galactose, these cells appeared to have a “mottled” appearance. On the other hand, WLPY010 and WLPY012 cells, whose parent strain was K699, had a uniformly fluorescent outer membrane under the same conditions.

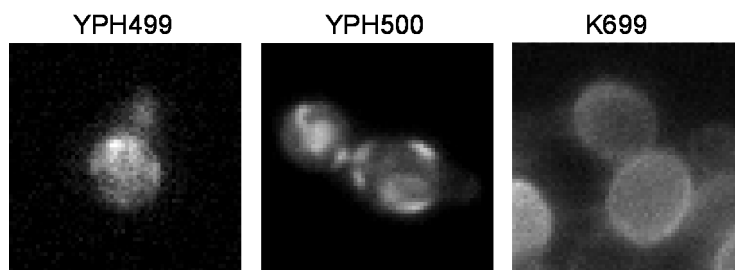


Figure 2.1: *GAL2* expression in *S. cerevisiae* YPH499, YPH500, and K699. Deficiencies in *GAL2* expression in YPH strains were verified using a yEVenus fusion. YPH cells exhibited a “mottled” appearance compared to the nearly uniform membrane distribution observed in K699 cells. This indicates poor localization of Gal2p to the cytosolic membrane and hence results in deficient utilization of galactose in the YPH lineage.

Table 2.3: Yeast variants created for this work. A “-” between gene names signifies a fusion product, where as a “:” indicates non-fused products that are coexpressed from the same promoter. Strains WLPY001 and WLPY003 were used in an unrelated project.

ID	Parent	Modifications	Marker
WLPY001	YPH500	GAL1:yEGFP-PEST (1x integrated)	TRP
WLPY002	YPH500	GAL1-yEGFP	HIS
WLPY003	YPH500	GAL1-yEGFP, gal80 Δ	HIS, URA
WLPY004	YPH500	GAL1-yCerulean (CFP)	HIS
WLPY005	YPH500	GAL1-yEVenus (YFP)	HIS
WLPY006	YPH499	GAL1-yCerulean (CFP)	HIS
WLPY007	YPH499	GAL1-yEVenus (YFP)	HIS
WLPY008	YPH499	GAL2-yEVenus (YFP)	HIS
WLPY009	YPH500	GAL2-yEVenus (YFP)	HIS
WLPY010	K699	GAL1-yEVenus (YFP)	HIS
WLPY011	K699	GAL1-yCerulean (CFP)	HIS
WLPY012	K699	GAL1-yCerulean, GAL2-yEVenus (CFP/YFP)	HIS, URA

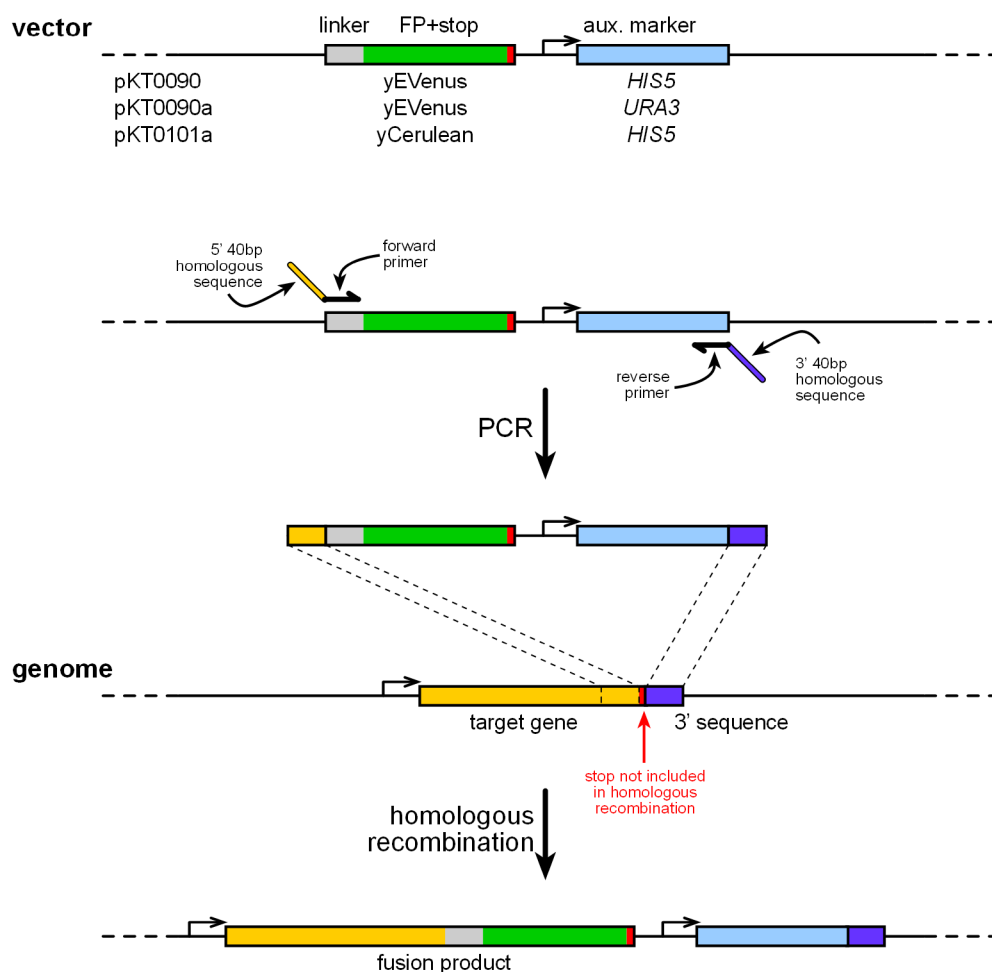


Figure 2.2: Yeast transformation using pKT derived fluorescent fusion protein vectors. Primers containing 40bp of genomic homology flanking the insert site are used to amplify the fusion reporter + selection marker from the plasmid vector. Introduction of the insert dna into the cell results in spontaneous homologous recombination targeted to the site specified by the initial primer sequences. Successful transformants exhibit both fluorescence under appropriate induction and illumination as well as auxotrophy for either histidine (HIS5), uracil (URA3), or both in the case of double transformants.

2.1.2 Cellular transformations

Fluorescent yeast variants were created using a well documented transformation protocol[51] which made use of *S. cerevisiae*'s inherent ability to merge homologous segments of DNA into its genome (Fig. 2.2). Parent strain cells were revived from frozen stock and grown on solid yeast-peptone/dextrose (YPD) + adenine (ade) media for two days at 30°C. A single colony was then selected and grown to saturation (OD₆₀₀ 4–6, approximately 1×10^9 cells/mL) in liquid YPD + ade media and diluted into 10mL of fresh media to an OD₆₀₀ 0.5 and further grown to late log (OD₆₀₀ 1–2). The culture was then washed in sterile 0.2 μ m filtered water and concentrated in 1mL of 0.1M Tris-EDTA buffered lithium acetate solution (LiAc buffer) at pH 8.0. Transforming DNA was created by PCR amplification from plasmids using primers that contained 40bp homologous recombination sites for the 3' end of yeast genomic *GAL1* and annealing sites that flanked the fluorescent protein to be fused, the fusion linker segment, and the auxotrophic selection marker. The PCR products were cleaned and purified using a PCR clean-up kit (Qiagen) and concentrated to approximately 500–2000ng/mL (yields varied based on PCR product size and efficiency). The transformation reaction mixture contained about 20–50ng of DNA, 1×10^7 cells (about 100 μ L of concentrated cell suspension), and 20 μ L of 10 μ g/mL single-stranded carrier DNA. The mixture volume was then brought to about 1mL with LiAc buffer with 40% (w/v) 3350kDa polyethylene glycol (PEG) and incubated for 30min at 30°C. To induce vector uptake, cells were heat shocked in a bench-top heating block at 42°C for 15min. The culture was immediately washed using room temperature sterile filtered water, and resuspended in approximately 150 μ L of fluid. Resuspended cultures were then plated onto selective synthetic-dropout (SD) media with appropriate selection supplements and incubated for

2–4 days at 30°C. Isolated colonies were picked, grown until saturation in selective media with 2% (w/v) glucose, and stored in 15% (w/v) glycerol at -80°C for later use.

While generally reliable, the above transformation protocol was surprisingly inefficient at transforming YPH cells, yielding at most 3–5 viable transformants per plate. After discussing this with Mike Ferry, a colleague in the lab, this was attributed to the relatively rare event of 40 bp of homologous sequence targeting the correct location within the genome. Instead, since the homologous recombination anchors were reasonably short, it was more likely that they targeted multiple undesired locations within the genome. Proof of this lay in the results of several attempts which yielded hundreds of “abortive” transformants per plate, noted by their exquisitely small colony size and inviability in selective liquid culture. To increase the efficiency of the protocol, ~400 bp segments of genomic DNA that flanked the desired insertion site were amplified in a secondary round of PCR. These “homology extenders” still incorporated the 40 bp homologous recombination anchors found on the original PCR primers. When incorporated into the transformation mixture in an equal ratio with vector amplified DNA, the homology extenders would combine with the vector insert using the same cellular recombination machinery needed to incorporate the construct into the genome. The new insert with longer homology targeting regions thus had a higher likelihood of recombining with the desired location. Transformations performed using this protocol were over 100 fold more efficient than using only 40 bp homologous sequences alone.

The cells used in this work were grown on several media conditions. All cultures used for experimental data collection were grown on SD medium. Not only did this media provide, slower, more controllable growth, but it also had a lower fluorescence background

than YP based medium. In total there were three basic media states that cells were grown in: neutral, non-inducing/non-repressing; inducing; and repressing. The neutral media condition was the simplest formulation, containing only SD (with appropriate auxotrophic selection supplements) and 2% raffinose, a sugar known to not interfere with either the galactose or glucose metabolic pathways[120]. Induction media contained varying concentrations of galactose with 2% raffinose. Here, raffinose was supplied as a growth supporting sugar. Cultures grown at low concentrations of galactose ($\leq 0.2\%$) without raffinose grew nearly 2 fold slower than cultures with raffinose present. In addition, higher cellular viability, and healthier morphology was observed in preliminary microfluidic studies of cells grown in raffinose as opposed to cells grown in its absence. Repressing media contained glucose, galactose, and 2% raffinose. Dynamic profiling of cellular expression primarily utilized inducing and repressing media. Thus, in the transition between these two media states, only the concentration of glucose varied, removing changes in gene expression due to changes in galactose concentration.

2.1.3 Steady-state expression characterization

The steady state induction and repression response of the galactose pathway in each strain was characterized using batch cultures and flow cytometry. This aided in the determination of realistic and dynamically interesting concentrations of galactose and glucose in subsequent experiments as well as the approximate growth rates of each strain under each media condition.

To generate induction curves, *S. cerevisiae* YPH499 and K699 cells containing yEVENUS fusions to Gal1p, (WLPY007 and WLPY010, respectively) were grown overnight in selective media containing only 2% (w/v) raffinose. Cells were then diluted in triplicate

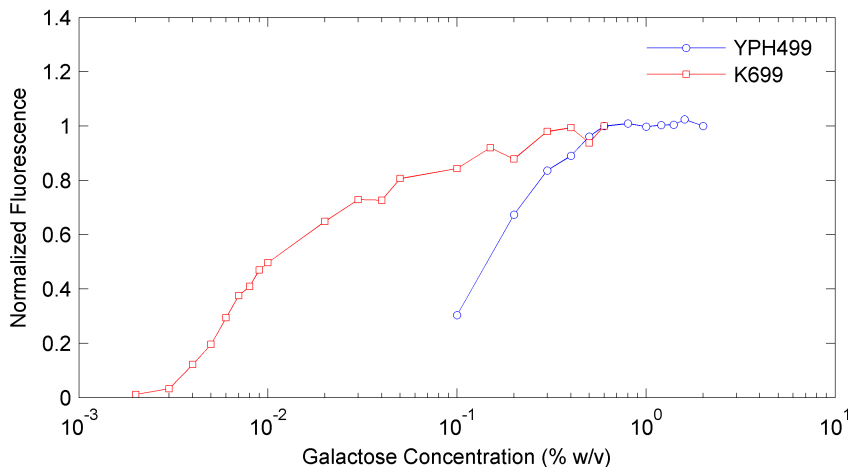


Figure 2.3: Steady-state galactose induction for *S. cerevisiae* YPH499 and K699. Note the dramatic difference in sensitivity between K699 and YPH499. YPH499 requires almost a 10-fold more galactose to reach the same induction level as K699.

to OD_{600} 0.1 into fresh media containing a range of galactose concentrations from 0–2.0% (w/v) (in addition to the original raffinose concentration) to initiate induction. Once each culture reached OD_{600} 0.6–0.7, they were analyzed for green/yellow fluorescence expression using a FACS Calibur flow cytometer (Beckton Dickinson). This way each culture would have the same number of generations prior to analysis, limiting possible cellular aging effects[3, 74]. The primary mode of the fluorescence histogram from 100,000 events was recorded as the expression level for each sample and values for fluorescence expression were normalized relative to the 2% galactose samples (Figure 2.3). YPH cells exhibited a fully induced state at approximately 1.0% (w/v) galactose, as noted by a plateau in their population weighted mean fluorescence, which was consistent with induction curves generated by Volfson et al.[109]. As expected the induction curve for K699 cells reached maximal induction at approximately 0.2% galactose, nearly an order of magnitude lower than YPH cells.

The generation of glucose repression curves for each yeast stain proceeded in a

similar fashion as the generation of induction curves. Again, YPH499 and K699 cells containing yE*Venus* fusions to Gal1p were grown to saturation overnight, however this time the media contained 2% (w/v) raffinose and 0.2% (w/v) galactose, producing a steady initial fluorescence state prior to glucose repression. Cells were then diluted in triplicate to OD₆₀₀ 0.1 into fresh media containing a range of glucose concentrations from 0–2.0% (w/v), in addition to the aforementioned galactose and raffinose concentrations, to initiate repression. Cultures were again grown to OD₆₀₀ 0.6–0.7 prior to fluorescence analysis on the flow cytometer. The data analysis was the same as the induction case and values for fluorescence expression were normalized relative to the 0% and 2% glucose samples (Figure 2.4). Here, the responses of both strains were nearly identical, saturating at approximately 0.25% glucose. The concentration of galactose was chosen for its dynamic significance and as a compromise towards the dynamic profiling of two kinetically different strains discussed in Chapter 3. At this level, YPH cells were only half induced while K699 cells were at the threshold of induction saturation. In both cases, cells of either strain would be as responsive as possible without incurring other unpredictable effects due to differing media states.

2.2 Microfluidic Devices

Several devices were created for this work starting with simple flow channels and ending with a complex platform capable of dynamic microenvironmental control and long duration growth and imaging of cellular populations with single-cell resolution. In doing so, many technical challenges were addressed, the most important being the maintenance of on-chip monolayer microcolonies of microbes.

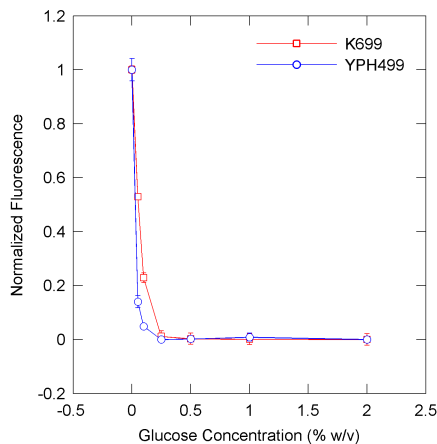


Figure 2.4: Steady-state glucose repression for *S. cerevisiae* YPH499 and K699. The repression response rapidly decays between 0% and 0.25% (w/v) glucose in both cell strains. Above 0.25%, fluorescence levels are negligible relative to the 2% glucose case, indicating full saturation of the response. Note that the two responses are nearly identical as opposed to the galactose induction curve.

2.2.1 Device fabrication

As previously mentioned, this work relied on PDMS rapid prototyping via replica molding for the creation of microfluidic devices. The techniques implemented here utilized a well documented [113, 112, 116] three part fabrication process involving photolithography, PDMS replica molding/soft-lithography, and chemical surface modification for bonding and fluidic sealing. More specialized fabrication methods, such as surface patterning using microcontact printing, were utilized on a device specific basis.

Devices were designed and prepared for photomask printing using AutoCAD 2005 (Autodesk Inc.). While other vector based illustration programs, such as Macromedia Freehand, Adobe Illustrator, or Corel Draw could also be utilized, it was important that drawings be output in AutoCAD DWG or DXF format, the standard used by all high resolution photomask printing establishments. While device drawings could also be output as encapsulated postscript (EPS) and converted using commercially available utilities

like LinkCAD, this process was prone to conversion errors and incurred additional design restraints making it more costly than using natively accepted formats.

Once finalized, designs were printed as patterns of clear and opaque regions at 20,000 DPI onto phototransparency film (Output City, Bandon OR) to be used as photomasks in the photolithographic process. At this resolution, the smallest lateral feature dimension was $7\mu\text{m}$. Smaller features could be printed, but could not be reproduced reliably. In addition, during the photolithographic process, clear patterns would typically expand by approximately $5\mu\text{m}$, and opaque patterns would be reduced by an equivalent amount due to the diffraction of light around masked regions. Thus, it was important to consider these effects and provide ample spacing between features to avoid “cross-fading” and merging when not desired. The absolute minimum spacing utilized for all devices in this work was $15\mu\text{m}$ and was only utilized in extreme cases. Overall, a more feasible and fault tolerant spacing was $30\mu\text{m}$, which also corresponded to the worst case scenario for photolithographic misalignment.

In the final printouts, patterned features printed as clear regions on a black opaque background were suitable for negative-tone photoresists such as SU-8, (Microchem Corp.), a photocurable epoxy that crosslinks when exposed to UV light. On the contrary, positive tone resists, such as SPR-1818 or AZ-100 (Shipley) required patterned features to be protected from UV exposure, and thus their photomasks contained opaque features on a clear background. The devices described here were fabricated using solely SU-8 patterned masters.

Printed phototransparencies were trimmed into individual masks and mounted onto clean $3'' \times 3'' \times 1/8''$ borosilicate glass plates (McMaster-Carr) by gluing only the cor-

ners of the transparency section to the glass using Loctite formula 495 superbonder. This bonding fluid was chosen over formula 490 because of its low viscosity, which allowed to for little distortion of the mask's "flatness" where the bonder was applied. In addition, the lower viscosity of the 495 formulation allow for better control of the amount of bonder used, and in all cases, as little as possible ($10\mu\text{L}$) was desired. The glass served as a rigid support for the mask so that it could be mounted to alignment equipment. In addition, since the photolithographic process required contacting of the photomask to the photopatternable resist film, the printed (emulsion) side was oriented facing outward to ensure appropriate pattern masking.

The most commonly used substrates for photolithography were glass or silicon wafers. I used silicon wafers in this work because they were readily available via our campus microfabrication facility and methods that utilized them were well established. Wafers were prepared by solvent cleaning with acetone, isopropanol, methanol, followed by a deionized water rinse, at 2000 rpm on a wafer spinner, and dehydrated for 5 min at 200°C using a contact hot-plate. These cleaning steps ensured that all manufacturing oils, moisture, and debris were removed from the substrate surface, all of which could detrimentally impede SU-8 surface adhesion. Adhesion promoting agents, such as hydroxymethylsilylazane (HDMS) or AP-100 (Microchem Corp.), were not required.

Photoresist was deposited onto silicon substrates by spin coating at various speeds to achieve a wide range of thicknesses ($0.5\text{--}300\ \mu\text{m}$). This allowed for the integration of multiple feature levels by repeatedly spin-coating, exposing to UV, and removing unexposed photoresist in SU-8 developer (Microchem Corp.) or propylene glycol methyl ether acetate (PGMEA, Sigma). Unlike positive tone resists, once UV crosslinked and thermally

cured, it was virtually impossible to overdevelop SU-8 patterns, thus previously patterned and developed topology was unaffected by repeated development steps. This allowed for stepwise fabrication of features with intermittent validation and adjustment of topology depth. Moreover, one could additively build up topographic features by omitting the development steps between layers, saving on costly photoresist material. Multilayer alignment was possible since patterned and thermally cured SU-8 was readily distinguishable from unpatterned resist films as long as the newly deposited film was less than five times thicker than previously patterned layer. This additive procedure was critical in reproducibly fabricating uniform film depths $\geq 300\mu\text{m}$ as opposed to using a single deposition of SU-8 100 or SU-8 2100 at low spin speeds ($\leq 1000\text{rpm}$).

Once a master mold was complete it was treated with vapor phase organo-silane (chlorotrimethylsilane, Sigma) for 2–5min. This formed a self-assembled methylsilane layer on the mold surface which aided in the release of PDMS replicas. Only one organo-silane treatment was necessary for each mold. Because organo-silane compounds are fairly corrosive, repeated treatment would result in degradation of SU-8 features. The most notable side effect of this was the spontaneous delamination of SU-8 features from the substrate surface due to degradation at the SU-8/silicon interface. Furthermore, excess silane from overexposure would transfer to PDMS replicas, reducing the effectiveness of subsequent bonding procedures and other PDMS surface modification steps.

Following silane deposition, degassed, liquid PDMS (Sylgard 184, Dow Corning), mixed 10 parts base to 1 part curing agent, was cast against the master mold to a depth of 5mm, and cured at 80°C for 1.5 hours. The PDMS replica, which now contained a negative image of the mold's topology, was then carefully peeled away from the master mold.

Fluidic access ports were bored through the replica using 20 gauge Luer stub adapters. The resultant hole was actually conical in shape, which would form a pressure tight seal with 23 gauge hypodermic needle tubing at the unpatterned surface of the PDMS replica. Larger ports, e.g. those required for channels that accommodated high flow ($\sim 1 \text{ mL min}^{-1}$), were punched using a 16 gauge Luer stub, which sealed against 18 gauge hypodermic tubing.

The replica was then thoroughly cleansed using a sonic bath, Scotch brand office tape, and a final isopropanol rinse to remove any debris prior to sectioning into individual PDMS chips. The devices were sealed with #1-1/2 coverslips by treating contacting surfaces to an oxygen plasma. This created an irreversibly bonded interface capable of withstanding up to 25psi of pressure[92, 118]. Alternatively, devices could be reversibly bonded to glass by first soaking the PDMS chips in dilute hydrochloric acid (0.01N) at 50°C for 2 hrs and annealing the chips to glass coverslips in a dry oven at 65–80°C for 10 hrs. Reversible seals could withstand only 10psi of pressure, however, this was typically sufficient for standard chip priming and cell loading procedures which required pressures at or above 3psi.

2.2.2 Confining microbial biofilms and microcolonies

Much of my initial microfluidic design work centered around the development of microdevices for use with bacterial cells. At the time, the bacteria *E. coli* was default host organism in the lab and it made sense to apply my microfluidic development accordingly. Up to this point in time, the only available tools for studying bacteria were simple microscopy, spectrophotometry/fluorimetry, and flow cytometry. However, due to the dynamic nature of the synthetic systems we were developing, we needed a method for studying time evolved fluorescence at the single-cell level.

Some of my early microfluidic device designs were for monitoring of microbial

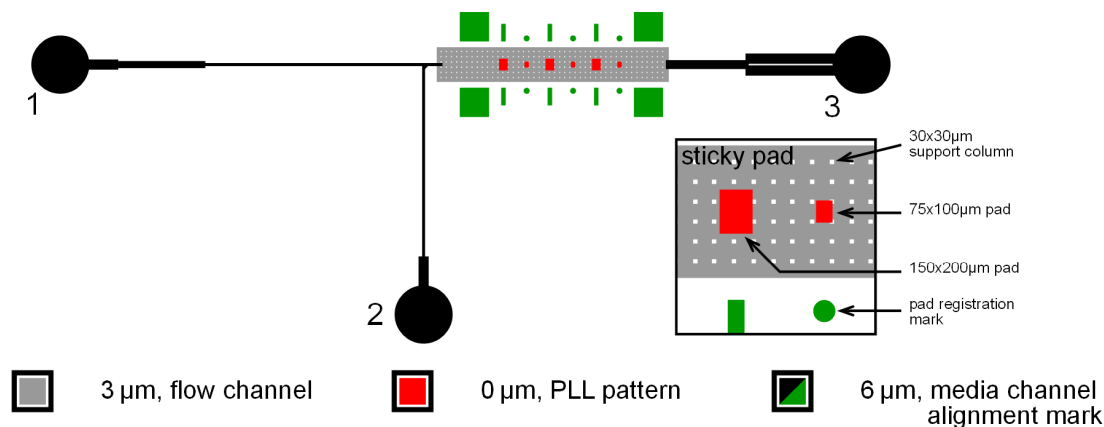


Figure 2.5: Schematic diagram of the “Sticky-Pad” device. Red regions within the gray flow channel are where poly-lysine “sticky-pads” are patterned using a micro-contact printing technique. Patterns are made using a small PDMS stamping block that contain columns with the same geometry as the red and green patterns shown. Cells are introduced into the device via port 3, and media via port 1. Port 2 is used as a common waste port and is positioned so that the probability of fouling of the media line (port 1) by cells is reduced.

biofilms in simple single channel systems that used patterned regions of poly(L-lysine) (PLL) on the sealing glass substrate. PLL, a relatively small polypeptide, exhibits a strong net positive charge at and above physiologic pH (7.2–7.4). A glass surface, on the other hand, has a net negative charge, as do the external surfaces of cells such as the bacteria *E. coli*. When patterned onto glass, the PLL formed “sticky-pads” that would capture cells allowed to settle over the patterned areas, retaining them electrostatically.

Using PLL to pattern *E. coli* had been used by several other studies investigating microbial biofilm development[23, 6], and represented an improvement over established agar pad technology[30] since the locations of biofilms could be precisely controlled. However, despite being laterally immobilized, biofilm growth in the vertical direction remained uninhibited, limiting experimental observation to the amount of time it would take to completely overgrow the region.

To overcome this, I utilized a 3µm deep microfluidic flow channel to subject the

captured cells to high levels of fluid shear stress (Figure 2.5). The flow channel was carefully aligned over 24×40 #1-1/2 coverslips patterned with PLL (70kDa) using a contact micro-printing technique (Figure 2.6)[89, 116]. Prior to patterning, the glass substrate was cleaned with solvent (acetone, isopropanol, and methanol) to remove any manufacturing process oils, and treated to an O₂ plasma making the surface more hydrophilic (as measured by an advancing water contact angle). To pattern the PLL, a plasma treated 3×5×5mm PDMS patterning stamp was wetted with 10μL of a 1mg/mL PLL solution and dried under a gentle stream of nitrogen gas. The stamp was then aligned pattern side down to the center of the cleaned coverslip using a stereo microscope and compressed with a small weight (approximately 20g per square centimeter). The PLL was allowed to adsorb to the glass substrate for 1hr at room temperature before removing the PDMS stamp and aligning the microfluidic channel. PLL patterns were verified by conjugating 1mM fluorescein isothiocyanate (FITC) in pH 9.0 sodium bicarbonate buffer and imaging using a epifluorescent microscope (Fig. 2.6).

Using a simple gravity system I could precisely adjust the flowrate through the device and linearly alter the fluid shear over the sticky-pads from 1.6-16dyn/cm². To test the device, it was loaded with *E. coli* cells that constitutively expressed GFP from a pZE21G high copy plasmid[22]. The cells immediately adhered to the sticky-pads in the chip and I was able to image their growth under 7.05dyn/cm² fluid shear for approximately 7hrs (Fig. 2.7b). The majority of the cells grew as desired, daughter cells that did not adhere to the sticky-pad surface were immediately washed away by the impinging fluid, restricting the colony to the attachment plane on the glass surface. Increasing the surrounding flow increased the amount of cellular detachment from pad, hence, colony size was controlled

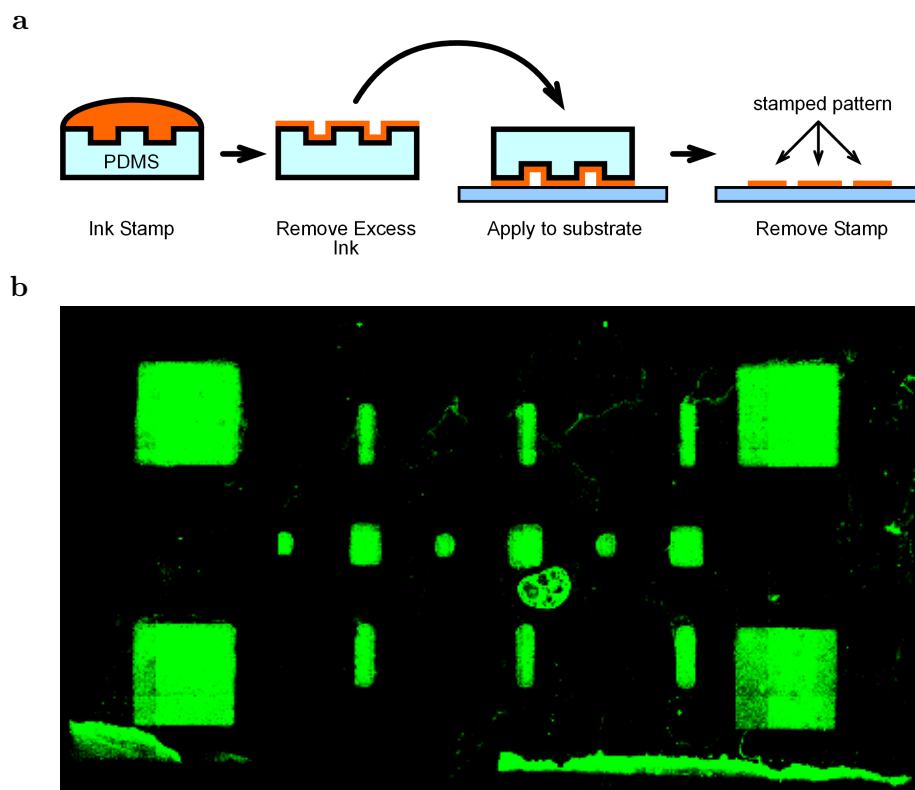


Figure 2.6: (a) Method for patterning “Sticky-Pads” using microcontact printing. (b) Sticky-pads visualized by FITC conjugation. Each large square in the corners of the pattern are approximately 1 mm square.

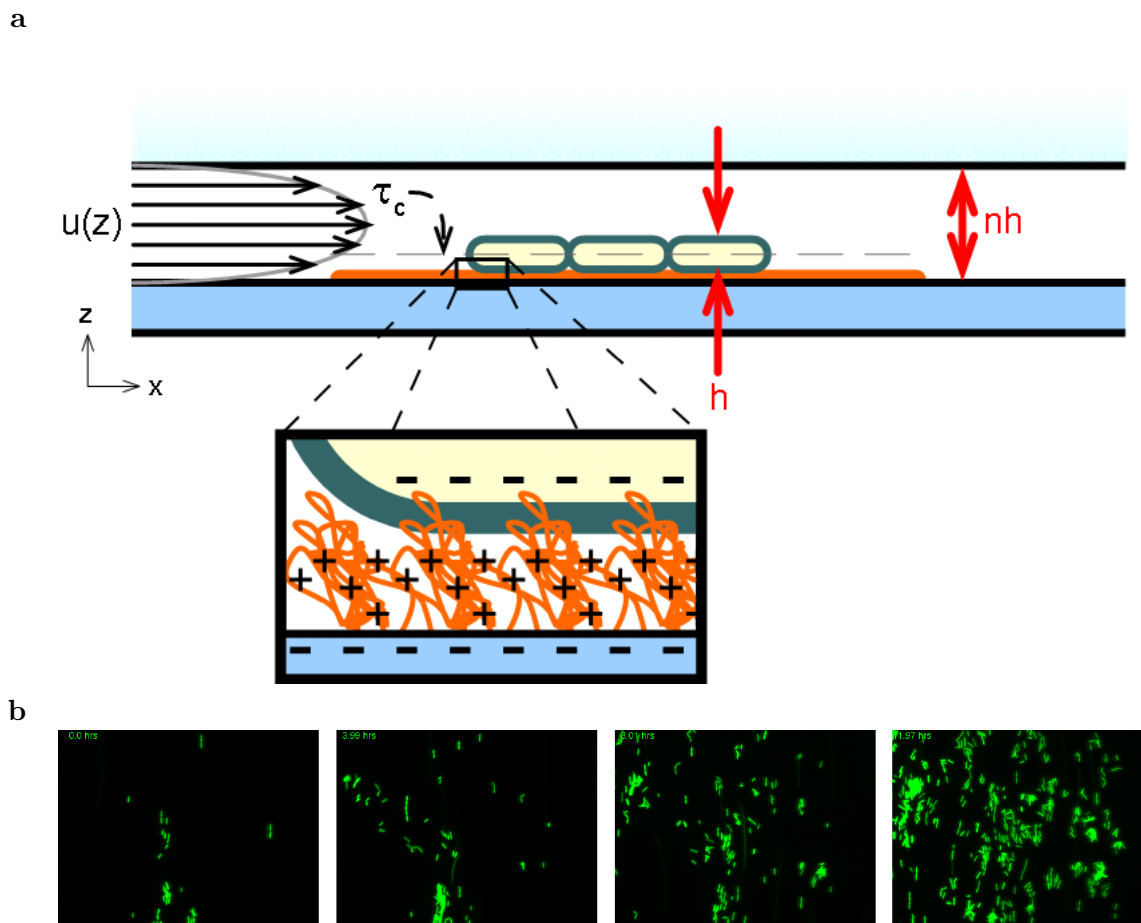


Figure 2.7: **(a)** The “Sticky-Pad” operational concept. Cells are adhered electrostatically to a glass substrate by patterned macromolecules (poly-lysine, orange) of opposite charge. The flow channel height is modified based on the nominal height of a cell adhered to a pad by a simple scaling factor n , which was optimized to a value of 3. In the case of the bacteria *E. coli*, $h = 1$, giving a channel height $3 \mu\text{m}$. The shear stress experienced by cells at the surface, τ_c , is a function of the velocity profile which is controlled primarily by channel height and bulk fluid flowrate. **(b)** Growth of *E. coli* cells constitutively expressing GFP on a sticky pad under approximately 7 dyn cm^{-2} of shear. Time between the first frame (left) and the last (right) is 8 hours. Fluid flows vertically, entering from the top of each frame.

via the superficial velocity of the fluid through the channel alone. Since nutrients were supplied by the bulk flow, each sticky-pad became a virtual culture vessel that maintained a static concentration of nutrients as well as a static population of cells, a microscale chemoturbidostat.

Although this was a promising initial result, subsequent testing revealed that inter-

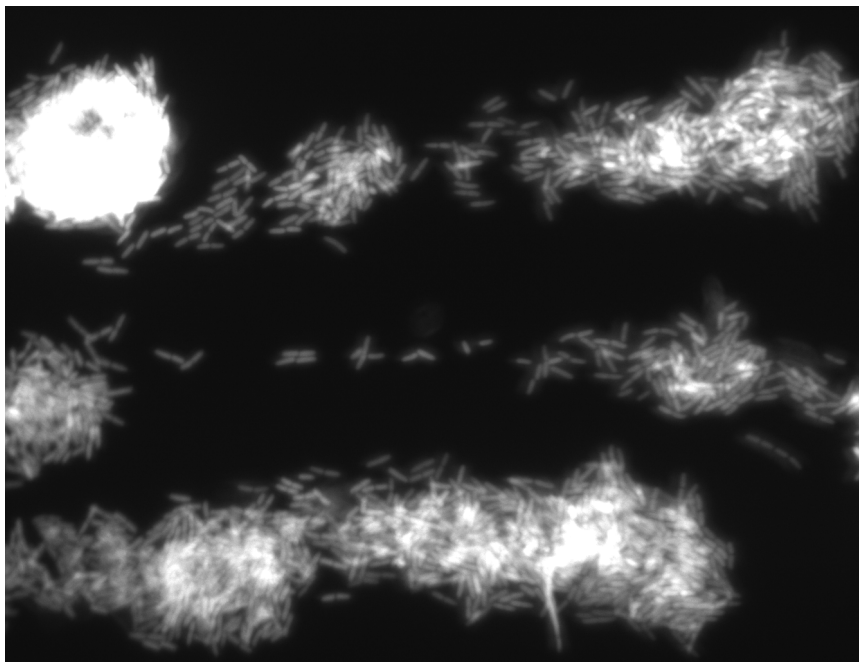


Figure 2.8: Colony comets on a “Sticky-Pad”. Growth of *E. coli* cells constitutively expressing GFP on a sticky pad after 12hrs under 10 dyn cm^{-2} . Flow is from right to left. Notice that cells in the “tails” are still distinguishable from neighbors, but cells in the “heads” are not due to vertical overgrowth.

cellular adhesion was far stronger than any shear forces that could be feasibly applied over the growing colony before disrupting electrostatic adhesion. Long duration growth under shear resulted in colony “comets” where a large mass of cells formed at the upstream end of a colony and gradually trailed to a few isolated cells at the downstream end (Fig. 2.8). In addition, closer examination of the time-lapse images revealed that cellular retention time within the colony was highly variable, ranging from completely static to almost immediate disappearance after division. On average, cellular retention time on the pad was approximately 0.5hrs, which was not acceptable for a quantitative expression analysis intended to last at least 12hrs, approximately 10-fold longer than the average bacterial cellular doubling time.

I next turned to a design that completely enclosed cells within a microchamber

similar to devices developed by Groisman et al.. These chambers were capable of trapping microbial cells (bacteria and yeast) while at the same time, providing a means for diffusive transport of nutrients within the microchamber. For my design, I modified the chemostatic microchamber so that it was a mere $1\mu\text{m}$ in depth, approximately the diameter of an *E. coli* cell, so that it would geometrically force the growing microcolony into a monolayer (Figure 2.9).

Called the eXtremely Low Chamber or XLC for short, initial developments proved problematic because of the selected chamber height and mechanical properties of PDMS. General design guidelines required that PDMS channels have a 30:1 width to height aspect ratio to withstand spontaneous feature collapse due to elastic stretching of the material. Despite adding support columns to the center of the microchambers to account for this restriction, many chambers still irreparably collapsed under the weight of the PDMS monolith above or due to normal chip handling. Attempts to harden the PDMS by baking unsealed chips at 150°C for 2hrs as reported by Groisman, resulted in only a slight improvement. An appropriate solution was to reduce the aspect ratio to 20:1. The same *E. coli* cells expressing high copy constitutive GFP were tested on-chip, and a microcolony after 5hrs of growth is shown in Figure 2.9.

While the new microchamber design was successful in maintaining a flat microcolony as desired, it had several drawbacks. Due to the constraints on chamber dimension, it would rapidly fill with cells growing at conditions comparable to batch culture (LB medium, 37°C). Even with minimal chamber seeding (3–5 cells), imaging sessions would last at most 8hrs. Beyond this point, cells would begin to spill out of the chamber through the diffusive feeding channels where they would foul adjacent nutrient delivery channels. Within the

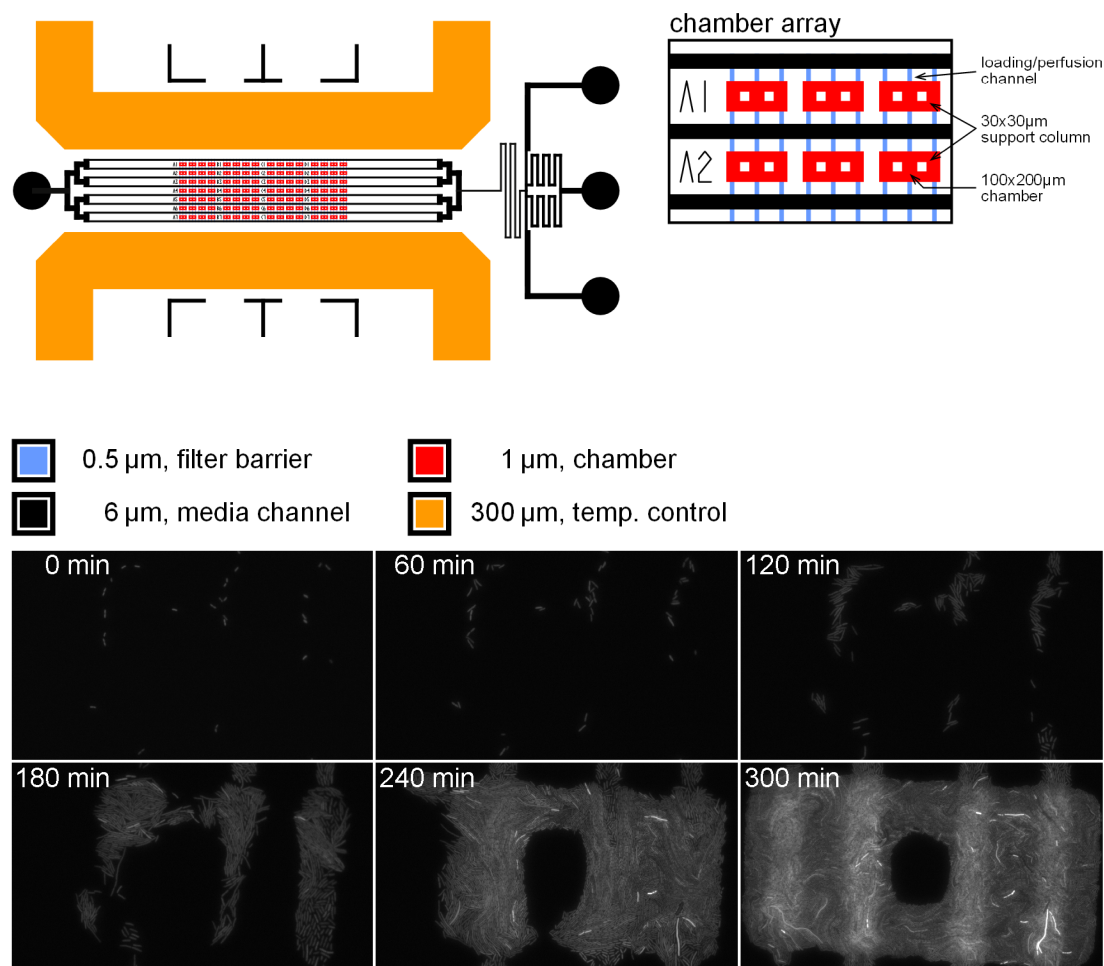


Figure 2.9: The XLC growth chamber array. (top) Device schematic. Chambers are arranged in banks of 7×4 to accommodate easy location of isolated chambers of interest. The chamber array sits downstream of a simple binary fluidic switch and between two independent thermal channels to facilitate step/pulse probing of cellular expression and thermal gradient studies, respectively. (bottom) Growth of *E. coli* constitutively expressing GFP over the course of 5 hours isothermally at 37°C in LB media supplemented with kanamycin. Notice that cells remain in a monolayer and identifiable from neighbors until the very last frame of the sequence when the chamber becomes confluent. Also notice that as the chamber fills, cells begin to escape through the diffusive feeding channels and into the primary flow channels immediately outside the chamber. The dark region in the center of the chamber is a support column and is approximately $30\mu\text{m}$ square.

chamber, cells would be so tightly packed that individual cell segmentation was difficult at the magnification used (40x with a field of view of approximately $150 \times 250 \mu\text{m}$).

Given both of these setbacks in the development of a bacterial growth and imaging device, it was clear that the technical challenges of imaging individual bacteria in a microdevice were too great, and a more “simple” model organism, from a microfluidic standpoint, was required. In addition, this would give me the opportunity to optimally design a growth chamber that had the trapping and turbidistatic/chemostatic abilities of both a sticky-pad device and a fully enclosed microchamber.

2.2.3 A quantitative, long-duration imaging platform

In collaboration with Scott Cookson, a colleague in the lab, we developed a microfluidic device specific to the analysis of the budding yeast *S. cerevisiae*[21]. We chose yeast for several reasons. Aside from being the de-facto eukaryotic model organism, they were much easier to handle in microfluidic systems. Unlike *E. coli*, *S. cerevisiae* were non-motile and had a more spherical morphology, which allowed for the use of simpler image segmentation algorithms. In addition, yeast cells were on the order of $4\text{--}5 \mu\text{m}$ in diameter, five times larger than *E. coli*, placing fewer structural constraints on the fabrication of microchambers. Moreover, the lab was slowly transitioning toward the use of *S. cerevisiae* as the default model organism because of the increased interest in complex regulatory behavior in eukaryotic gene expression.

Still, if left to grow unconstrained, yeast would grow into three dimensional aggregates, making quantification of single-cell fluorescence expression states difficult without specialized techniques such as confocal or deconvolution microscopy. While impressively precise, such techniques presented a major constraint on the temporal resolution of acqui-

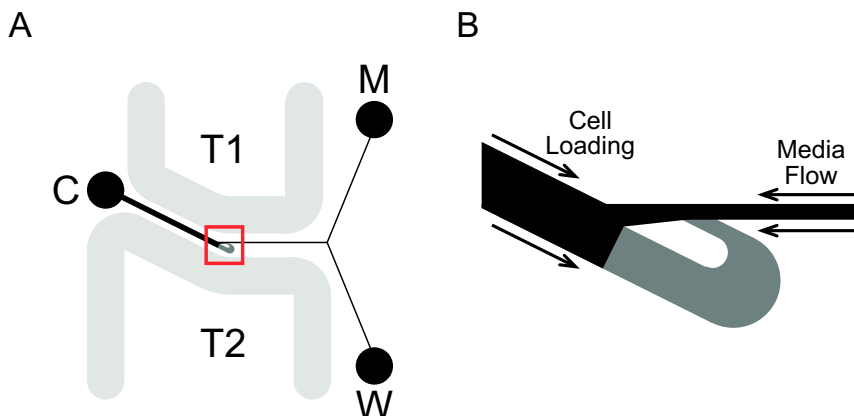


Figure 2.10: A schematic overview of the telta microchemostat, or $T\mu C$, a microfluidic device optimized for long-term growth of cells in a monolayer. (a) Overall view of the $T\mu C$. Three separate ports for loading cells (C), media perfusion (M), and waste collection (W). On-chip temperature is controlled using heated water running through large channels (T1 and T2) positioned near the growth chamber. (b) An enlarged view of the growth chamber. The height of the chamber (dark grey) is customized based on cell type. The flow channels (black) are 2–3 times higher than the trapping region, depending on the desired flow difference between the main channel and the side-arm. Cells are loaded by flowing culture toward the growth chamber from the cell inlet port. Once cells are trapped, flow is reversed within the adjacent channel, perfusing fresh media from the media port through the device.

sition, especially for genetic processes with time scales on the order of minutes. Acquisition speed could be improved, but at the sacrifice of spatial resolution, limiting analysis to small fields of cells.

The microfluidic chamber Scott and I developed utilized a similar flat chamber concept to the fully enclosed bacterial chambers I had developed earlier. This allowed for the use of more temporally efficient, wide-field epifluorescent imaging for fluorescence quantification. The immediate results of this technique were dramatic. As shown in Fig. 2.17, a colony of yeast cells growing without constraint proliferate in multiple spatial dimensions, making the extraction of single-cell fluorescence data difficult, while constrained cells are distributed in a planar fashion with distinct boundaries between neighbors.

In addition to being $4\mu m$ deep, instead of the $1\mu m$ depth used for *E. coli*, the growth chamber in this new device had a few other unique features. The chamber was

actually a specialized loop structure resembling a subunit of Tesla's valvular conduit[97] which allowed for unidirectional flow[29] for the loading and trapping of cells. The entrance to the "trapping loop" had a large ($140\mu\text{m}\times 4\mu\text{m}$) open structure which allowed for easy cell entry and escape as well as unrestricted nutrient influx since it presented a large mass transport area. The free escape of cells was actually advantageous to the use of the device for extended observation sessions, since it allowed the chamber to passively maintain a constant cell population within without overcrowding the imaging field. Once cells exited the chamber they were immediately swept away to a waste reservoir by a fast moving fluid stream, similar to the operation of the sticky-pad device. The free escape of cells, coupled with rapid clearance of the nearby fluidic channel allowed the device to be operated for durations on the order of 50–100 cell divisions without the fear of fouling by cellular overgrowth. The chamber was named the $T\mu C$ for Tesla micro-chemostat because of its unique geometry and its ability to maintain a colony of cells under both chemostatic and turbidistatic conditions.

Because of the size of the $T\mu C$ growth chamber and its extended operation time, it was critical to ensure that nutrients were adequately provided to cells in the portions of the chamber farthest away from the entrance. To do this I analyzed the transport characteristics of the $T\mu C$ both theoretically and experimentally.

The key parameters involved were the diffusion coefficient of each nutrient species and the advective velocity through the chamber. Most important was the determination of the magnitude of the advective velocity since it profoundly affected the efficiency at which the device was capable of establishing an optimal growth environment for cells. To do so, the characteristics of the transport process were first analyzed by deriving a closed

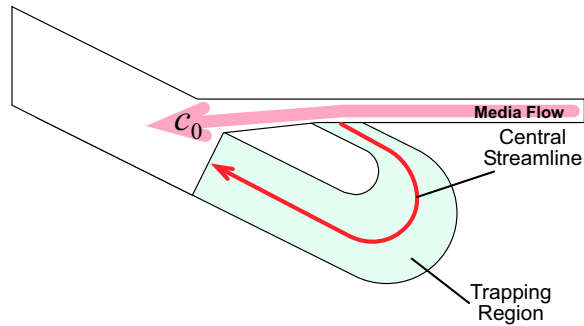


Figure 2.11: T μ C Advection/Diffusion Analysis Schematic

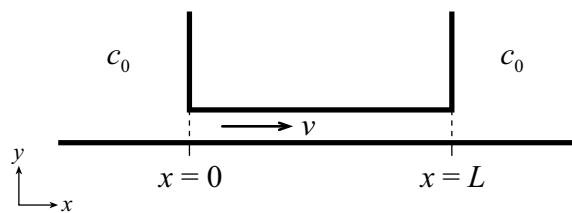


Figure 2.12: 1D representation of the T μ C with diffusive and advective transport

form solution of the time dependent chemical concentration profile throughout the growth chamber.

The chemical species mass balance for unsteady diffusion with advection is,

$$\frac{\partial c}{\partial t} + \vec{v} \cdot \vec{\nabla} c = \mathcal{D} \nabla^2 c \quad (2.1)$$

Since geometric and physical parameters restrict on-chip flow to the laminar regime, analysis was isolated to the central streamline of flow within the chamber, shown in Figure 2.11. This assumed that components of velocity and spatial gradients in concentration normal to this streamline were negligible relative to those tangential to the direction of flow, reducing the physical process to the 1D system shown in Figure 2.12, simplifying Eqn. 2.1 to,

$$\frac{\partial c}{\partial t} + V_0 \frac{\partial c}{\partial x} = \mathcal{D} \frac{\partial^2 c}{\partial x^2} \quad (2.2)$$

with the following boundary and initial conditions,

$$c(0, t) = c_0, \quad t > 0 \quad (2.3)$$

$$c(L, t) = c_0, \quad t > 0 \quad (2.4)$$

$$c(x, 0) = 0, \quad 0 < x < L \quad (2.5)$$

Here, V_0 is the average channel velocity, \mathcal{D} is the molecular diffusivity of the nutrient chemical species, and c_0 is both the steady-state and boundary supported concentration.

Equations 2.2 to 2.5 can be further simplified by substituting the following dimensionless parameters,

$$C = \frac{c - c_0}{c_0} \quad \xi = \frac{x}{L} \quad \tau = \frac{\mathcal{D}}{L^2} t \quad (2.6)$$

giving,

$$\frac{\partial C}{\partial \tau} + N_{\text{Pe}} \frac{\partial C}{\partial \xi} = \frac{\partial^2 C}{\partial \xi^2} \quad (2.7)$$

$$C(0, \tau) = 0, \quad \tau > 0 \quad (2.8)$$

$$C(1, \tau) = 0, \quad \tau > 0 \quad (2.9)$$

$$C(\xi, 0) = -1, \quad 0 < \xi < 1 \quad (2.10)$$

where,

$$N_{\text{Pe}} = \frac{V_0 L}{\mathcal{D}} \quad (2.11)$$

and the Peclet Number, a dimensionless ratio of advective transport versus diffusive transport. Since the system is linear with respect to C with homogeneous boundary conditions, it is easily solved using the method of separation of variables[39], yielding the closed form solution,

$$C(\xi, \tau) = \exp\left(\frac{N_{\text{Pe}}}{2} \xi\right) \sum_{n=1}^{\infty} A_n \sin(n\pi \xi) \exp(-\lambda_n \tau) \quad (2.12)$$

or,

$$c(x, t) = c_0 \left\{ \exp\left(\frac{N_{\text{Pe}}}{2L} x\right) \sum_{n=1}^{\infty} A_n \sin\left(\frac{n\pi}{L} x\right) \exp\left(-\lambda_n \frac{\mathcal{D}}{L^2} t\right) + 1 \right\} \quad (2.13)$$

where,

$$\lambda_n = \frac{1}{4}[(2n\pi)^2 + N_{\text{Pe}}^2] \quad (2.14)$$

$$A_n = 4 \exp\left(-\frac{N_{\text{Pe}}}{2}\right) \left\{ \frac{2n\pi \cos(n\pi) + N_{\text{Pe}} \sin(n\pi) - 2n\pi \exp\left(\frac{N_{\text{Pe}}}{2}\right)}{N_{\text{Pe}}^2 + (2n\pi)^2} \right\} \quad (2.15)$$

To complement the analytical solution, experiments testing the real world mass transport within the chamber were performed using red fluorescent dyes of varying molecular weight. To do this, the chamber was first wetted with water entering from the cell loading port and imaged to provide a baseline condition. Water with fluorescent dye was then introduced from the media port and the chamber was imaged every 5 seconds for 1.5 hours.

The image sequence was processed by removing background intensity and correcting for non-uniform illumination using a flat-field frame. The sequence was then smoothed

using a 5 pixel square gaussian kernel to remove remaining image noise. A quadratic interpolating spline was fit to 6-10 points manually chosen along the central streamline within the trapping region and subdivided into 100 individual points. For each image in the sequence, path profiles were generated by recording the mean pixel intensity over a line $10\mu\text{m}$ in length by 1 pixel wide normal to the spline as the fluorescence value at each path point. Experimental concentration was assumed to be linearly proportional to fluorescence signal, and was normalized by the steady-state value.

Initial experimental profiles used 10kDa dextran conjugated Rhodamine B isothiocyanate whose size was equivalent to small polypeptide signaling factors such as α_1 -mating factor. The diffusion constant of the dye was estimated to be $7.99 \times 10^{-7} \text{cm}^2 \text{s}^{-1}$ using the Stokes-Einstein equation for large diffusing particles[12],

$$\mathcal{D}_{AB} = \frac{k_b T}{6\pi\mu_B s_A} \quad (2.16)$$

where \mathcal{D}_{AB} is the diffusion coefficient of solute A in solvent B, k_b is Boltzman's constant, T is the ambient temperature, μ_B is the dynamic viscosity of solvent B, and s_A is the molecular stokes radius of solute A. For dextrans, the value of s_A is determined using a previously reported correlation[107] based on molecular weight

$$s_A = 0.488 (MW)^{0.437} \quad (2.17)$$

The estimated diffusion coefficient value, along with a value of $1\mu\text{m/s}$ for the advection velocity V_0 , and a path length, L , of $570\mu\text{m}$ (approx. the path length analyzed in the experimental data), was used to generate the analytical simulation shown in Figure 2.13, providing a $N_{\text{Pe}} = 7.125$. Analytical simulation data is plotted as $C'(\xi, t) = C(\xi, t) + 1$

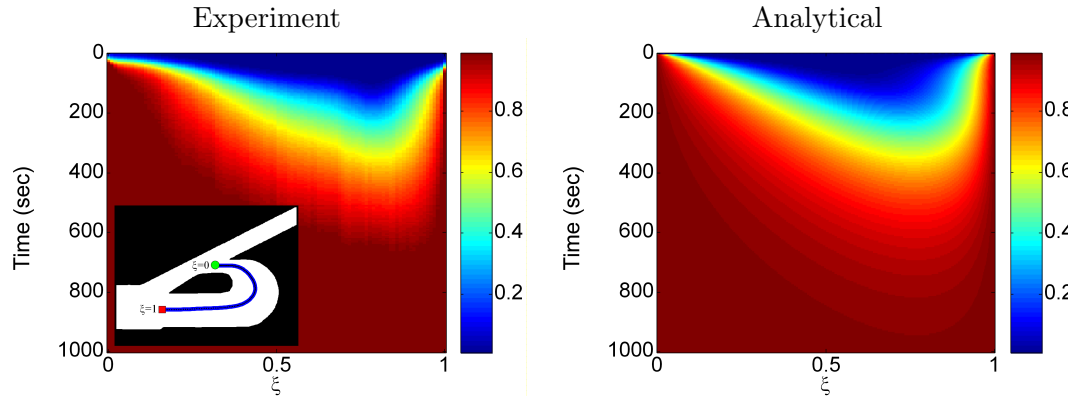


Figure 2.13: Comparison of experimental data collected using 10kDa Dextran conjugated Rhodamine B isothiocyanate along the central streamline within the trapping region to the analytical solution of 1D diffusion with advection for large molecule transport. The inset in the experimental data shows the path analyzed (blue line) starting from the green circle, and ending at the red square.

calculated from a 100 term fourier series. The analytical results were in good agreement with the experimental data, supporting the existence of creeping flow on the order of $1\mu\text{ms}^{-1}$ within the trapping region of the device under nominal operational conditions.

Asymmetry in the concentration profile over ξ was primarily attributed to advective transport. Closer inspection of the analytical solution revealed that as $N_{\text{Pe}} \rightarrow 0$, Eqns. 2.12-2.15 degenerate to the solution for simple 1D unsteady diffusion, causing the physical process to become spatially symmetric about $\xi = 0.5$. Moreover, the speed of the transport process was determined to be highly dependant on the value of N_{Pe} , as seen in Figure 2.14.

To further test the validity of the model, a similar experiment was performed using a smaller molecular weight dye, Sulforhodamine 101, which is approximately the same size as the nutrient components of growth media. For these smaller, polar, molecular species, a different form of the Stokes-Einstein equation must be used,

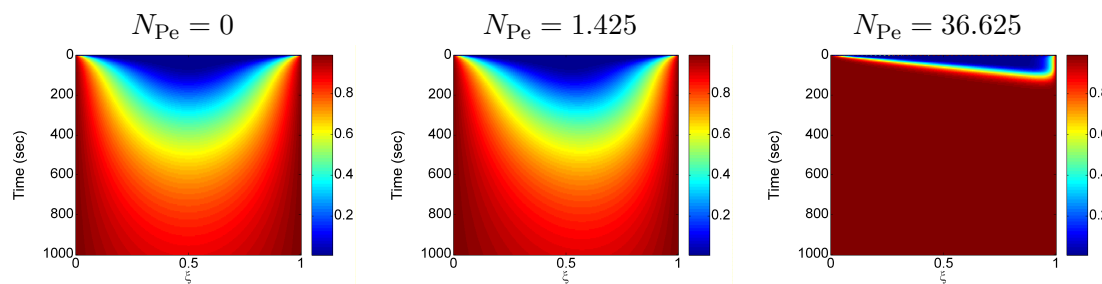


Figure 2.14: (Left to Right) Analytical analysis of the time evolution of 1D concentration profiles for advective velocities of $0\mu\text{m/s}$, $0.2\mu\text{m/s}$, and $5\mu\text{m/s}$.

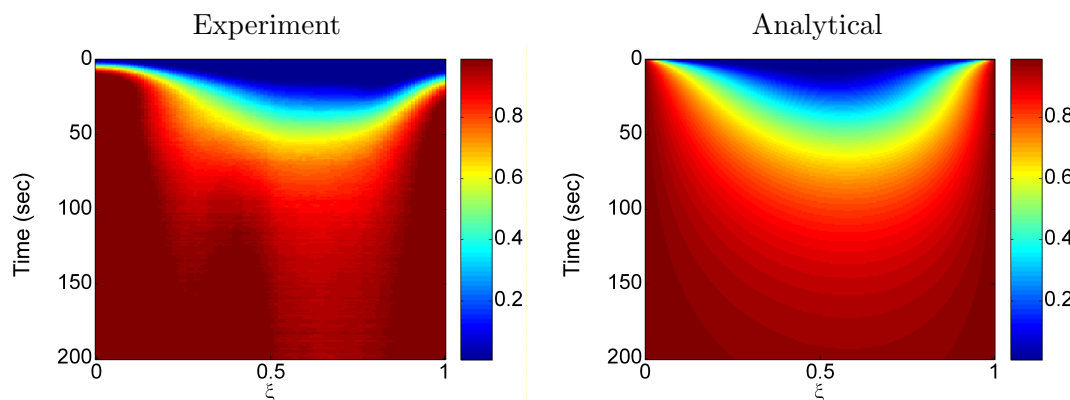


Figure 2.15: Comparison of experimental data using Sulforhodamine 101 to an analytical solution of small molecular transport by 1D diffusion with advection. The inset in the experimental data shows the path analyzed (blue line) starting from the green circle, and ending at the red square.

$$\mathcal{D}_{AB} = \frac{k_b T}{4\pi\mu_B s_A} \quad (2.18)$$

which accounts for fluid slip at the molecular surface. Estimates of the stokes radius for these species is typically made by analysis of the bond lengths between constituent atoms, however, for generalized order of magnitude estimates, the diffusion constant for small chemical species is assumed to be on the order of $6 \times 10^{-6} \text{cm}^2 \text{s}^{-1}$. Using this value for \mathcal{D} resulted in a $N_{Pe} = 1.615$. Experimental and simulation data analyzed as before was still in good agreement (Figure 2.15).

As a final case, it was important to understand how the presence of cells affected the chamber transport processes. Once a cell population filled the chamber, the fluidic resistance through the chamber would reach its maximum value due to the increased fluidic tortuosity and decreased hydraulic flow radius. Under these circumstances, the transport process becomes similar to fluid flow and mass transport through a tightly packed bed. Within the governing transport equations, this manifests itself as scale factors for the advective and diffusive terms.

Changes to the advective component of transport are made by estimating the superficial velocity using the Blake-Kozeny equation[12],

$$V_0 = \frac{\Delta P}{L} \frac{d_P^2}{150\mu} \frac{\epsilon^3}{(1 - \epsilon)^2} \quad (2.19)$$

where, ΔP is the pressure drop over the channel, d_P is the diameter of the packing particle, μ is the fluid dynamic viscosity, and ϵ is the channel void volume fraction. Because diffusive transport is dependent upon the available mass transfer area, the diffusive term in eqn. 2.7 must be scaled accordingly. Thus, an open area fraction,

$$\phi = \frac{\text{area}_{\text{open}}}{\text{area}_{\text{total}}} \quad (2.20)$$

is used such that,

$$\mathcal{D}_{\text{eff}} = \phi \mathcal{D} \quad (2.21)$$

Assuming that cells are perfect spheres with tight hexagonal packing, the value of $V_0 \approx 10 \times 10^{-6} \mu\text{m/s}$ and $\phi = 1 - \frac{\pi}{4}$ reducing the value of N_{Pe} to $\approx 4 \times 10^{-5}$. Under these conditions, the transport process is considered purely diffusive. To verify this, a Sulforho-

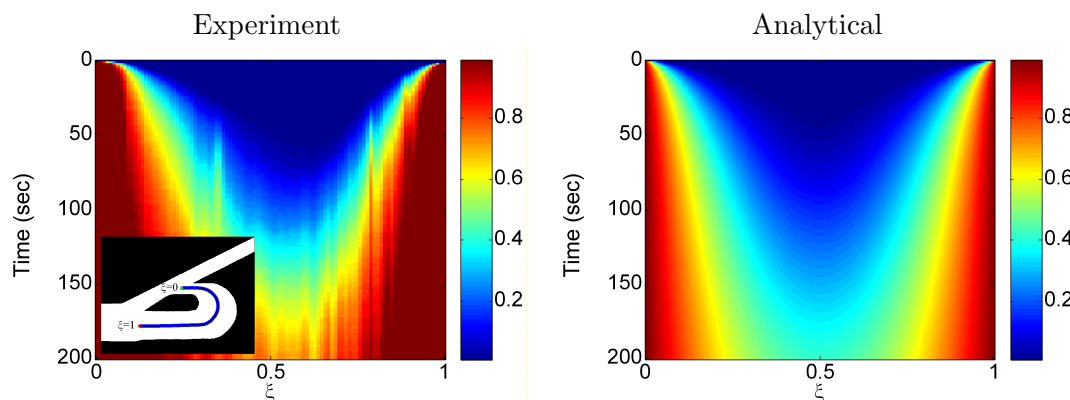


Figure 2.16: Comparison of experimental data collected from a full chamber using Sulforhodamine 101 to an analytical solution of small molecular transport by 1D diffusion with advection through a packed bed. The inset in the experimental data shows the path analyzed (blue line) starting from the green circle, and ending at the red square.

damine 101 was again used to visualize nutrient transport into a chamber completely packed with cells (Figure 2.16). Again, the analytical simulation was in good agreement with experimental data, confirming primarily diffusive transport (negligible advective velocity) through a completely full chamber.

From the above analysis, the characteristic transport times to the furthest interior regions of the chamber for the primarily diffusive case were found to be on the order of 5–10min (Fig. 2.16). This was assumed to be negligible compared to the nutrient uptake rate of cells and was verified experimentally by measuring the cellular growth rate at various distances from the chamber entrance.

2.2.4 Controllable dynamic microenvironments

In work that I did upon first joining Dr. Hasty’s group at UCSD, I designed and tested a chemo-thermo gradient intended to explore the dynamics of the co-repressive toggle, an extension of studies performed by Gardner et al.[34] and Kobayshi et al.[58]. The

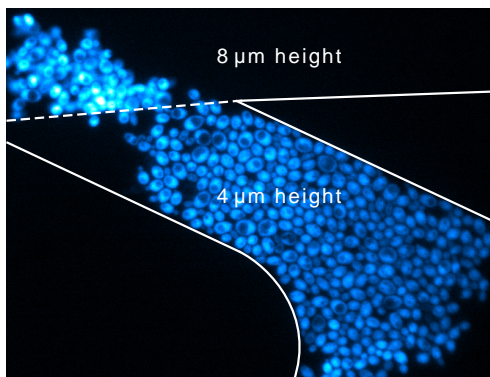


Figure 2.17: Cells growing within the $T_{\mu}C$. The chamber height is $4\mu m$ while the neighboring channel is $8\mu m$ high. Cells that grew outside of the chamber are difficult to individually distinguish, in contrast to cells within the chamber where cell-cell boundaries are clearly defined

device design featured a static gradient mixer described in previously published work by Dertinger et al.[27]. Later, it became apparent that a more dynamic control of the gradient profile would yield more information from the gene expression system such as kinetic rates of induction and repression leading to the distinct states of the toggle. This led to the idea of a dynamic gradient generator which coupled a laminar flow focusing module to the input of a chemical gradient mixer.

In this dynamic gradient device (Fig. 2.18; see Appendix D, Section D.4), a high concentration stream was focused by laminar flow using low concentration streams and directed to one side of an output channel. The output channel flow was then split and fed as inputs to a gradient mixer with one half acting as the “low” concentration input and the other as the “high” concentration input. Controlling which side of the output channel the high concentration stream was directed toward ultimately controlled both the magnitude and direction of the resultant gradient profile. Thus, the output profile from

the device could exist in one of four states: hi-hi, hi-lo, lo-hi, and lo-lo. In addition, the profiles were not restricted to discrete states, but could be continuously varied by gradual movement of the focused high concentration stream across the output channel. Observing cells as the surround media changed according to these profile changes would have provided valuable information regarding how the concentration of inducers affected both the rates of gene expression and magnitude of expression noise that governed the stable states of the regulatory network. While the project was eventually abandoned due to numerous technical challenges and the publication of a similar device[68], the idea that a simple fluidic switch based on laminar fluid focusing could produce continuously varying chemical profiles was not.

While the T μ C was useful for passive monitoring of cell growth and native, unperturbed, gene expression, it was limited by its inability to monitor systems that required external stimuli to reveal gene expression dynamics. Moreover, the study of the dynamics of many native and *de novo* gene regulatory networks can only be done with external stimuli that either initialize the system, or continually drive its dynamics. For example, the genetic toggle presented by Gardner et al. required both chemical and thermal external stimuli to actuate system changes. Similarly, the repressillator, a synthetic oscillator presented by Elowitz and Leibler[30], required a pulse of the chemical inducer IPTG to initialize the system.

To accommodate these needs, I designed a device which allowed for controllable, yet highly dynamic, microenvironments within the growth and imaging chamber. The creation of these microenvironments required the generation of concentration waveforms of chemical stimulants (e.g. inducer or repressor molecules) and their uniform transmission

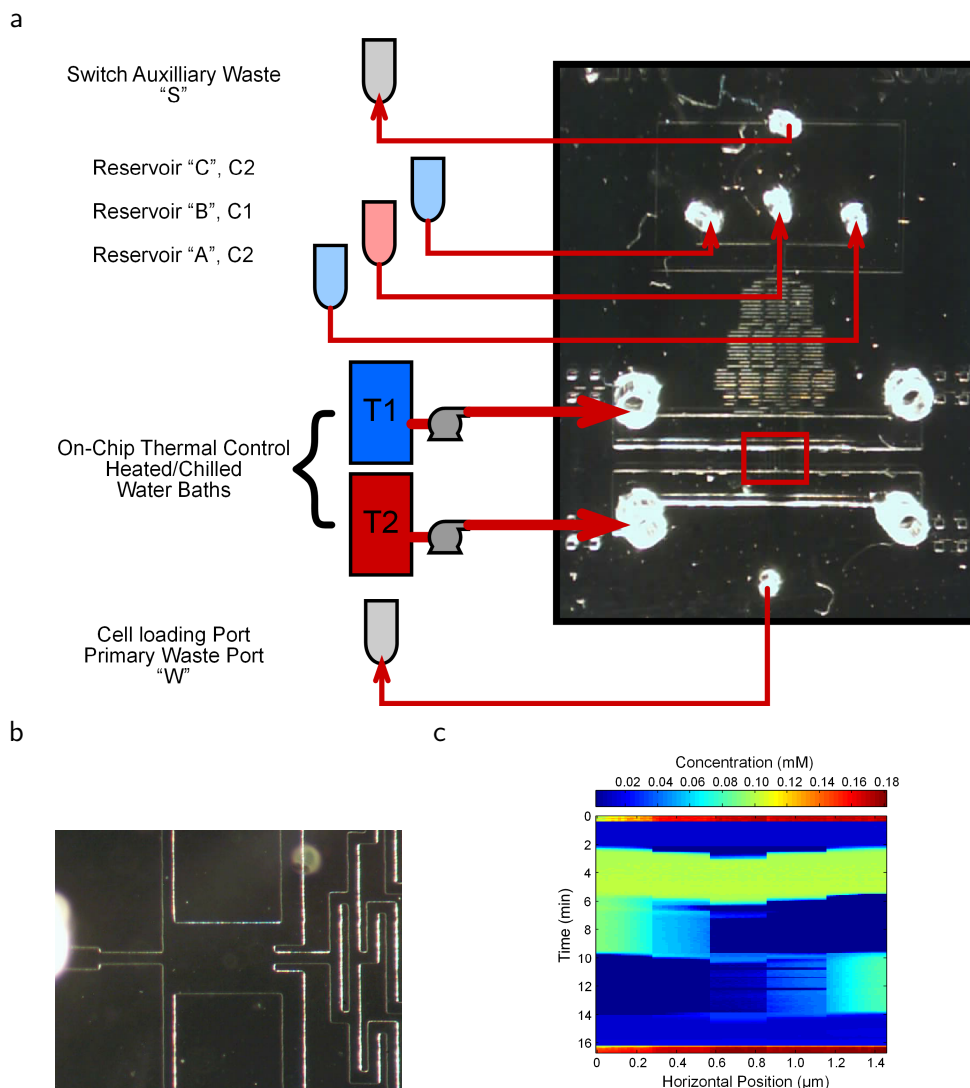


Figure 2.18: A dynamically controlled gradient profile device. The device (a) features a linear gradient mixer downstream of a fluidic input switch (b). As shown in (c), the switch allows for the generation of four distinct gradient profiles: lo-lo, hi-lo, lo-hi, and hi-hi at time scales on the order of 1 minute.

throughout the growth chamber. This new device used three flow networks for loading cells, generating microenvironmental waveforms, and controlling on-chip temperature.

To drive concentration waveforms, the device incorporated a fluidic media switch, developed using my prior experience with the dynamic gradient device and published literature[36, 65], (Fig. 2.19c) and a series of four chaotic advection mixing units[96]. Down-

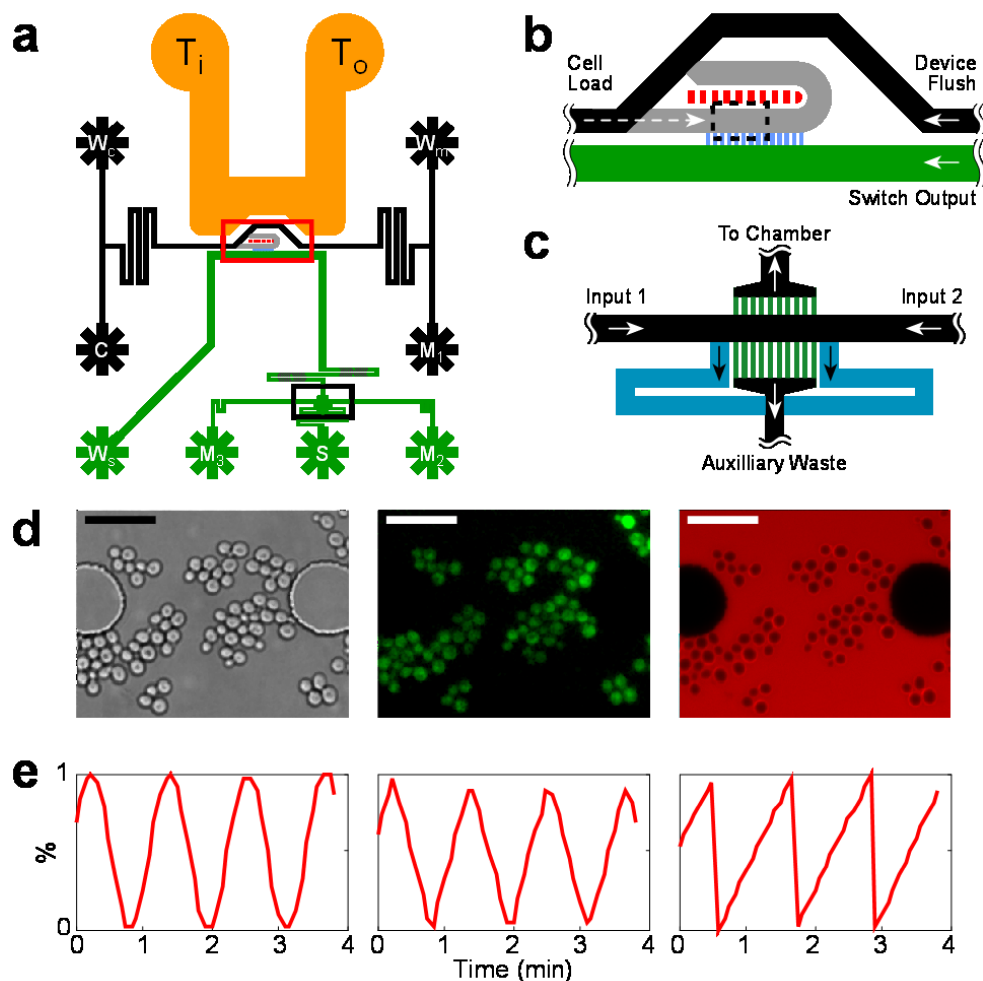


Figure 2.19: The microfluidic device utilized in this study. **(a)** Overall device schematic. The device uses three flow networks for loading cells (middle, black), generating microenvironmental waveforms (bottom, green), and controlling on-chip temperature (top, orange). The monolayer imaging chamber (center, red boxed region) is shown in more detail in panel (b). The fluidic switch (bottom, black boxed region) is shown in more detail in panel (c). **(b)** Magnified view of the imaging chamber (grey). The chamber is coupled to the switch output channel via multiple $1\ \mu\text{m}$ tall channels. These “feeding” channels also aid in loading cells by acting as a filtration barrier. The boxed region indicates a typical field of view. **(c)** Magnified view of the fluidic switch. Parallel guide channels (green) aid in producing a linear range of mixing ratios between two impinging input flows. Bypass channels (blue) collect and redirect excess flow to a waste reservoir. **(d)** Representative images of cells growing within the imaging chamber. From left to right, brightfield, green fluorescence, and red fluorescence. Scale bar is $25\ \mu\text{m}$ in length. Large circular regions are support posts in the chamber. **(e)** Sample waveforms generated by the fluidic switching/mixing system. From left to right, normalized fluorescence of tracer dye for sine, triangle, and sawtooth waves measured within the imaging chamber.

stream of the waveform generator, was the monolayer growth chamber, similar to that used in the T μ C device design[21] (Fig. 2.19b).

The fluidic switch, the primary element of the waveform generator, was operated by laminar interface guidance, a method similar to laminar flow focusing. For example, to “switch” a flow stream, the laminar interface between input flows was guided across an output channel by carefully adjusting the ratio of flow rates between the inputs (Fig. 2.20). Previously published fluidic switches were designed to operate in a binary fashion, either fully “ON” or fully “OFF”. The switch on my device was specifically designed to generate the intermediate mixing ratios required for accurate waveform production. To do this the device included flow guide channels in the output stream. These channels utilized the Coanda effect[45] which caused fluid flow to stick to the channel surfaces. This significantly improved the precision and accuracy of the output mixing ratios by improving control of the location of the laminar interface in the output channel. With these channels in place, the position of the interface could be linearly positioned by the relative pressure difference between the two reservoirs, with a difference of zero resulting in 50% output mixture. The result was a continuous linear range of output mixtures from 100% input 1 to 100% input 2 (Fig. 2.21).

Downstream of the waveform generator, a series of microchannels was used to couple the growth chamber to waveform output, allowing for the rapid transport of media throughout the observation area. These channels also provided a filtration barrier between the growth chamber and the waveform system, limiting overgrowth of cells into the waveform output channel. Lastly, the geometry and distribution of the feeding channels was designed to uniformly convect nutrients throughout the chamber. This allowed for the exchange of

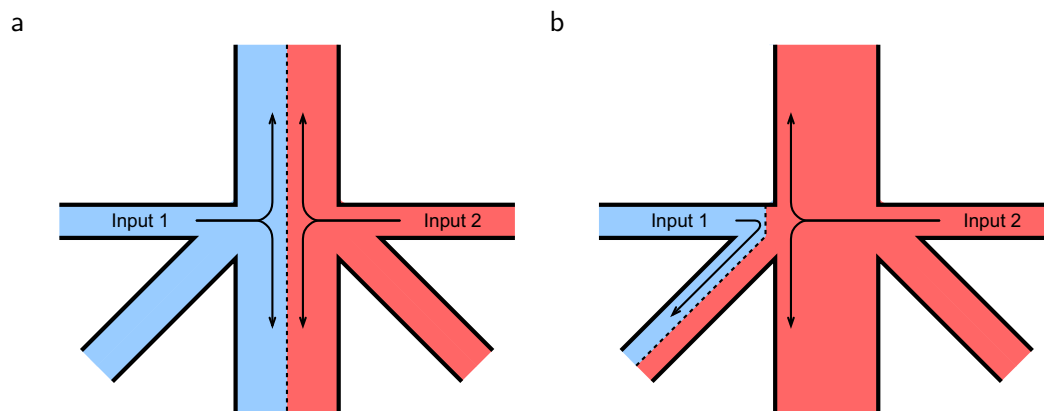


Figure 2.20: Cartoon depiction of laminar interface guidance

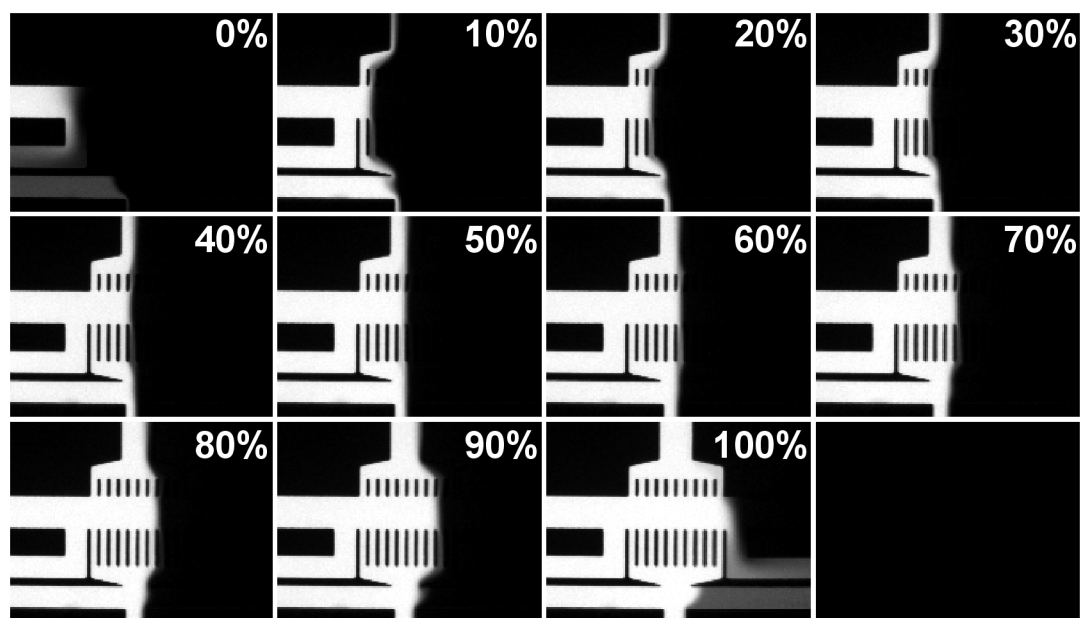


Figure 2.21: Images of graded mixing output from the on-chip switch with output flow guide channels. Input streams are water with and without a red fluorescent tracer dye (sulforhodamine 101).

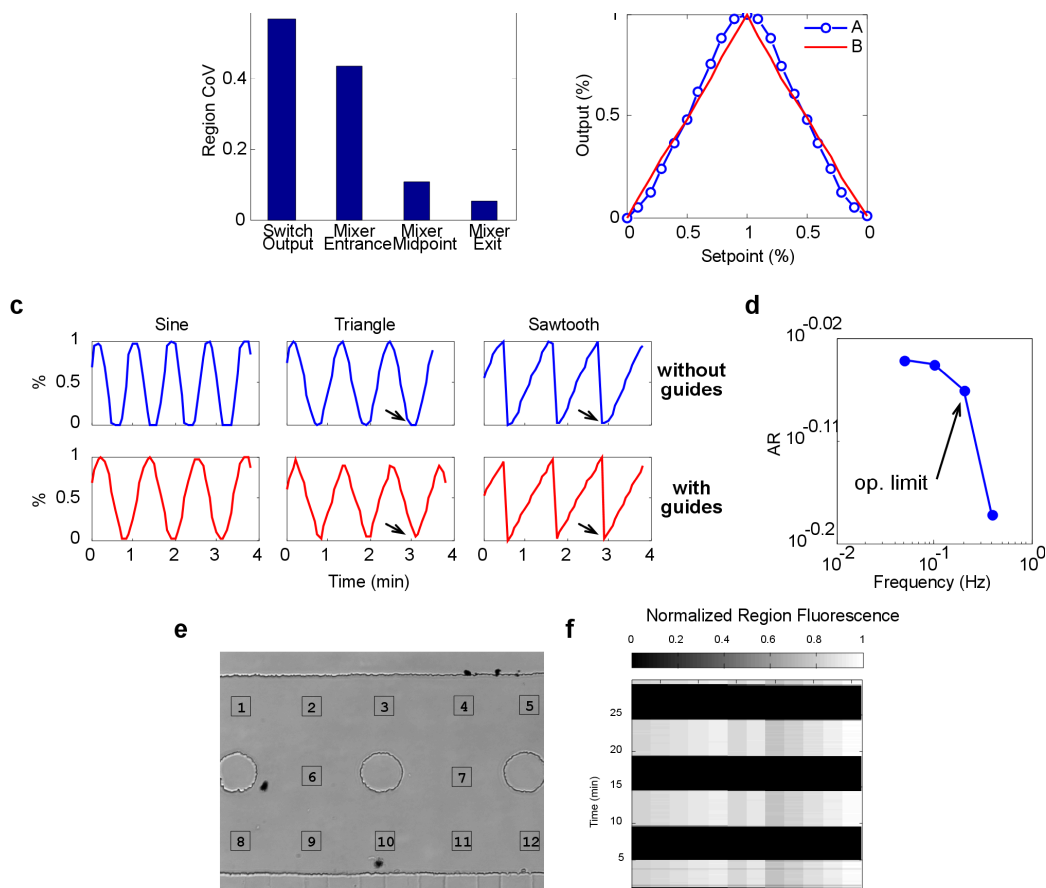


Figure 2.22: Data from various tests used to characterize on-chip waveform generation. (a) Concentration distribution (profile uniformity) at various sections of the device measured as regional coefficient of variation. (b) Output response curves for each device. Curve A (—○—) shows the response of a switch without output flow guides. The response has 20% deviations at the extremes of the output range, in contrast to a device with flow guides, curve B (—), which exhibits an ideal linear response. (c) Various waveforms generated by the system using devices without (top row) and with (bottom row) flow guides at a nominal frequency of 8.3 mHz. Shown here are sinusoidal, triangle, and sawtooth waves created by the system, measured using a red fluorescent tracer at the exit of the mixer train. Arrows highlight noticeable differences in waveform quality. (d) Frequency response analysis of the system showing amplitude ratio for frequencies ranging from 0.05 Hz to 0.4 Hz. The operational threshold is indicated by the sharp drop in the amplitude ratio at 0.2 Hz. (e) Map of $15 \times 15 \mu\text{m}^2$ regions of interest (ROIs) within a $40\times$ field of view in the growth chamber (black box in Figure 1a) used to measure concentration uniformity. (f) Spatio-temporal profile of fluorescent tracer concentration within the growth chamber with environmental changes driven at a nominal frequency of ~ 3 mHz. Deviations between regions are relatively small (~ 5 – 10%) and are deemed negligible. Temporal uniformity is ideal.

chamber contents to occur on the order of 0.5–1 min, an order of magnitude faster than the diffusive transport method used by the T μ C.

Operational performance of the waveform generator was evaluated by tracking the

distribution of a red fluorescent dye (sulforhodamine 101, Sigma) at several points within the device. These locations included the switch output channel and several points along the mixing train with $75 \times 75 \mu\text{m}^2$ regions analyzed at each location. Measurements within the mixing train were taken at the entrance, middle (after two mixing units), and exit regions. Measurement of dye distribution was separated into two parameters: concentration and uniformity, measured as regional mean fluorescence and coefficient of variation (C_V), respectively. Ideally, uniform regions should exhibit a C_V of zero. Under experimental conditions, this value was approximately 0.05–0.06, due primarily to image acquisition noise. As desired, The chaotic mixers smoothed the channel concentration profile from a sharp step function to a uniform distribution across the output channel. This was indicated by an approximate 4-fold drop in region C_V between switch and mixer outlets (Fig. 2.22a).

The linear range of output mixtures was tested by collecting images from the mixer exit for a rising/falling stair-step routine with 10-second holds at each step for system equilibration. Results of this test (Fig. 2.22b) show a nearly ideal linear response for a device with flow guide channels (curve B). A device fabricated without these guide channels (curve A) suffered from 20% deviations at the upper and lower extremes of the output range. Fig. 2.22c shows sinusoidal, triangle, and sawtooth waveforms generated by each device. Output deviations in the device without flow guides were especially noticeable in the triangle and sawtooth waveforms, resulting in a rounding of what should be sharp transitions in measured dye concentration. Under appropriate operating conditions, the device was capable of attaining a maximum output frequency of 0.2 Hz. This was determined using a frequency response analysis from 0.05 Hz to 0.4 Hz, which indicated a sharp drop in the output amplitude ratio at the maximal frequency (Fig. 2.22d).

Transport speed and uniformity within the growth chamber was tested by pulsing sulforhodamine at 3 mHz and observing the concentration at 12 separate locations within a $40\times$ field (Figs. 2.22e and 2.22f). Results of this test show that although deviations exist, they are relatively small, $\sim 5\text{--}10\%$ of the maximum measured value. In addition, the temporal uniformity within the chamber was as desired, e.g. changes in local concentrations at different subregions occurred practically simultaneously. The speed of the system was important for precise temporal control of the microenvironment, presenting a useful means for studying environmental noise effects on gene network dynamics. This could be done by incorporating additional noise components into the output waveforms, or by replicating a purely “noisy” environment by generating a random-step waveform.

As in the case of the $T\mu C$, it was important to verify that nutrients were adequately reaching cells throughout all portions of the chamber when full confluence was reached. More importantly in the case of this new device, nutrients needed to be transported at a rate many times faster than the nominal cellular growth rate so that it could be assumed that all cells were experiencing external perturbations simultaneously. To do this the chamber was allowed to fill with yeast cells via overnight on-chip growth. Sulforhodamine 101 was then pulsed through the chamber and images were acquired at 5 minute intervals. The results (Fig. 2.23) show adequate ($> 90\%$) perfusion of a fully confluent chamber within 10min, which is approximately one tenth the nominal doubling time for yeast.

This new device was named the $T^2\mu C$ for Temporal Tesla micro-Chemostat, owing to its resemblance to the original $T\mu C$, and its ability to generate temporally variable growth environments within the imaging chamber. In addition, there were several significant design improvements in the $T^2\mu C$ over the original $T\mu C$.

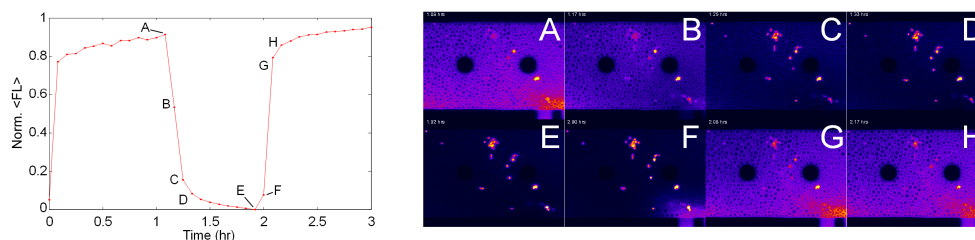


Figure 2.23: Nutrient transport in a confluent $T^2\mu C$ growth chamber. Mean intensity of a red fluorescent tracer traced through a series of images as it convects through a $T^2\mu C$ growth chamber full of yeast cells. Images are spaced 5 minutes apart and the driving signal is a simple square wave. Full transitions, A-F and E-H are twenty minutes in length, however 90% of the desired fluorescence state is reached within 10 minutes, approximately one tenth the nominal doubling time of yeast.

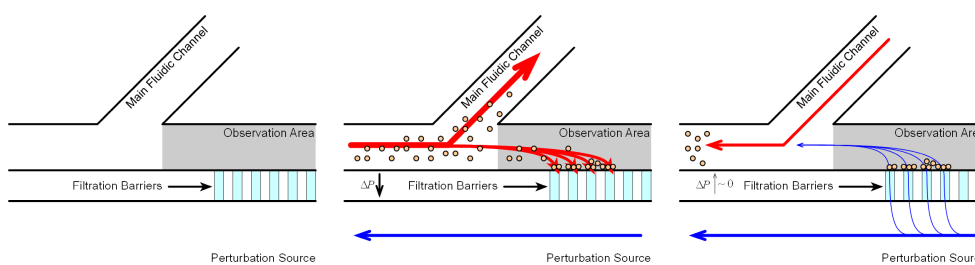


Figure 2.24: Loading of the $T^2\mu C$. Cells are first primed into the loading segment immediately outside of the imaging chamber and then directly injected inward using applied pressure. Because the height of the filter/feeding channels are only $1\text{--}2\mu\text{m}$ in depth, cells are retained at the wall of the chamber that is coupled to the dynamic media delivery channel. Once a satisfactory seeding population is achieved, flow within the chamber is reversed, supplying media from the dynamic media delivery channel, and also redistributing cells clustered at the coupled chamber wall.

First, the cell handling fluidic network was designed with an “h-cross” network rather than the “t-junction” used by the $T\mu C$. This provided a far more robust method of limiting device fouling by providing dedicated waste streams for both the cell suspension and loading media inlet. Flow between these reservoirs and their corresponding waste outlets would never cease, allowing no opportunity for cells to settle into the laminar flow boundary layer at the floors fluidic channels where they would be nearly impossible to remove. The design also provided a cell loading segment that could be primed with a defined number of cells and subsequently injected into the imaging chamber.

Second, because the growth chamber of the $T^2\mu C$ specifically allowed for convective

transport, loading the device relied on direct filtration of cells from the loading media (Fig. 2.24), a more controllable method than the passive capture procedure utilized by the T μ C. Upon priming the loading segment with cells, on-chip pressure was modified so that flow from the loading segment was directly injected into the imaging chamber. Cells would then be retained by the feeding channels, fabricated to 1–2 μ m depth, while fluid was allowed to pass through to the coupled media switching channel. Lastly, on-chip pressure was readjusted so that fluid from the switching channel was redirected back through the chamber and out toward the cell loading segment. In most cases, the filtration step would cluster cells at the wall of the imaging chamber occupied by feeding channels. This reversal of flow helped to redistribute cells through out the growth chamber, aiding in subsequent image processing and object extraction and quantification.

2.3 Microscopy

2.3.1 System description

All imaging was performed on a Nikon Diaphot TMD advanced research grade inverted epifluorescence microscope outfitted with a Hamamatsu Orca-ER cooled CCD firewire camera. To control light exposure, the system was fitted with Uniblitz VS35 high speed shutters (Vincent Associates) on both the fluorescence and transmitted light sources. Multiple fluorescence channels were acquired using a “Sedat” illumination configuration which utilized individual excitation and emission filter pairs (Chroma Inc.) mounted on filter wheels in the acquisition light path. An EXFO X-Cite 120 with a power attenuation iris was used as a fluorescent light source and a standard Nikon HMX-4 halogen lamp was used as a transmission light source. To reduce photobleaching and phototoxicity during

fluorescence exposure, lamp power was reduced to 12.5% using the built-in attenuation iris. This allowed for ≥ 1 second exposures without detrimentally affecting cellular viability and fluorescence reporter brightness.

All image acquisition hardware was controlled by a graphical user interface developed in LabVIEW (National Instruments, Inc) and designed solely for multichannel imaging and area/pattern scanning. Image acquisition was controlled via NI-IMAQ for IEEE 1394 camera drivers. For each image, the camera was externally triggered using a 10V TTL signal generated by a NI PCI-6021 multifunction digital and analog input/output board. Shutters were controlled via the control output bits on a standard PC parallel port and timed accordingly to the camera acquisition. All other automated system elements (XYZ motion and filter wheel positioning) was controlled via RS-232 command sequences sent to the Proscan-II controller.

2.3.2 Vibration isolation

Special care was taken to isolate the specimen stage from vibrations, as it dramatically affected on-chip flow control and acquired image quality. The entire microscope system was assembled on top of a vibration dampening table (Newark) to isolate it from common vibration sources such as user entry and exit from the imaging room. Even with this precaution, the light source shutters remained a major source of system vibration, the worst being that used for transmitted light illumination. Through trial and error testing it was determined that a shutter open delay of 60ms was required for fluorescence imaging, to allow for vibrations to dampen through the microscope body to undetectable levels. The vibrations due to the transmitted shutter required significantly longer to quell, about 3s, when solidly coupled to the transmitted light boom, which was an unfeasible amount of de-

lay time for high-speed imaging applications. Thus the solution was to completely decouple the shutter from the system, and instead mount it on an independent stand with the light path aligned by hand.

2.3.3 Reduction of thermally induced drift in focal plane

Images were acquired using an Olympus 40 \times magnification, 1.00NA oil immersion objective. Because this objective thermally coupled the microfluidic device to the microscope body, the surrounding air was maintained at $25\pm 0.1^\circ\text{C}$. To further control on chip temperature, heated water was circulated through channels, $1\text{mm}\times 300\mu\text{m}$ in cross section. These channels were placed within $250\mu\text{m}$ of the observed growth regions of the device and on-chip temperature was assumed to be the average of the input and output flow temperature within 1°C accuracy.

2.3.4 Autofocus method

Due to the extended duration of the imaging runs, an active and automated routine was used to maintain the imaged focal plane, to account for z-drift in the specimen due to remaining thermal fluctuations in the room. A dedicated camera system was mounted onto one of the binocular eyepiece ports of the microscope. Video signal from this camera was sent to a Prior Scientific Proscan-II AF enabled XYZ stage controller for processing and focal plane adjustment and autofocus routines were run prior to the acquisition transmitted imaging channels. These routines typically took only 1–2 seconds to complete, scanning over a range of approximately $10\mu\text{m}$. There were no AF routines run during subsequent fluorescence channels. Instead, calibrated z-offsets required for sharp fluorescence images were used. This minimized the amount of time required to acquire a channel “stack” and

exposure to the light source used for autofocus illumination.

During initial system development, attempts were made to use the contrast signal from cells in view for autofocus plane acquisition. Unfortunately, this was not adequate under brightfield imaging conditions, and compatible phase contrast optics were not available for our optical train. Non-fluorescent latex beads were used as a second attempt, and while they provided a crisp contrast signal, a trace of the focus scores through the desired scan range displayed two local maxima. Since the automated focusing system was hardware based, there was no method to programmatically select the more appropriate maxima without drastically reducing temporal efficiency of the imaging system. While, the use of fluorescent beads provided a more ideal focal score trace, controllably distributing the beads into the imaging field in a repeatable manner proved to be a significant challenge. To provide a constant and consistent object for the autofocus system to scan, a repeating pattern of air filled chambers $25 \times 50 \times 4 \mu\text{m}$ in dimension was placed along the length of the growth chamber. Although intended for brightfield/phase-contrast imaging, the most reliable autofocus signal was achieved using fluorescent illumination using the DsRed excitation filter (558 nm) and no emission filter (Fig. 2.25). When scanning through the focal planes, the optimal plane consistently fell at the level where the PDMS came into contact with the coverglass, which conveniently coincided with the optimal focal plane for cellular imaging. In addition, the location of the autofocus scan pattern, approximately $100\text{--}150 \mu\text{m}$ away from the imaging field, further reduced unintended fluorescence exposure which could lead to photobleaching and phototoxicity.

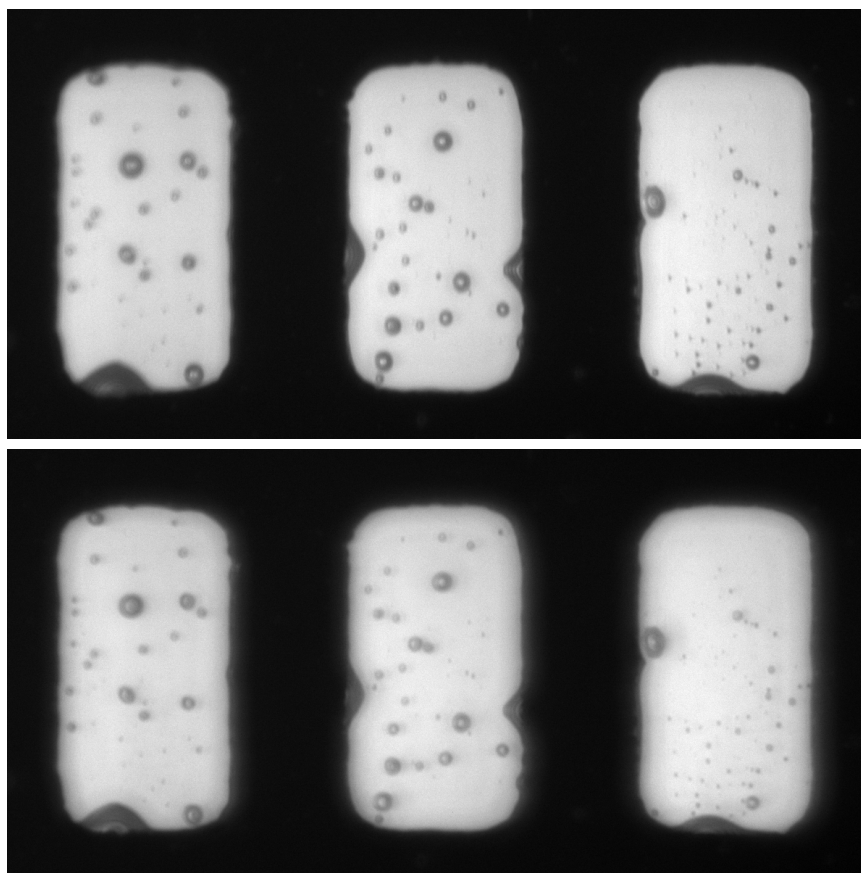


Figure 2.25: An reliable autofocus pattern using dry air chambers and fluorescent illumination. Images shown were acquired using 558 nm illumination with no emission filtration, imaged at 40 \times magnification. Each of the isolated dry chambers is 25 \times 50 μ m in dimension. The top image is in focus while the bottom image is not, observed as a “haze” around feature edges.

2.3.5 Image processing and analysis

Images were analyzed using a variety of software tools. Simple image processing and analysis methods were performed using the free software utility ImageJ (NIH). More complex routines, such as single cell segmentation and trajectory analysis, were performed using routines written in Matlab. To generate single cell trajectory data, individual cells were located by segmenting brightfield images. This was done using a sequence of grey scale reconstructive morphological operations to remove random image noise, remove spurious markings that could interfere with later processing steps, and enhance cell-cell boundaries. The image was then binarized using a basic threshold for objects statistically “brighter” than the image background. The binary image was further cleaned to fill holes in objects and remove isolated pixels. To identify individual objects, the binary image was processed using a distance transform followed by a watershed algorithm. The resultant watershed lines were then used to enhance existing object borders, and separate touching objects. Each individual object was then labeled with a unique ID and the intensity statistics for corresponding pixel regions were measured in the red and green/yellow fluorescence images. Object data was stored in human readable text files for each time point within the image sequence. To track objects through the sequence, the object data was recompiled into a large data array and fed into a tracking algorithm previously used in colloidal particle studies[24]. Running fully autonomously the above method was capable of segmenting and tracking individual cells with a 75–80% accuracy rate. Manual editing of the segmentation mask used for object labeling and validation/linking of object trajectories increased this rate to 90–95%. On average, it took 30min of computer processing and up to 8–10hrs of manual correction and validation to generate high quality trajectories for 100–200 cells

from an image set with 300 image time points. For studies that did not require single-cell resolution, individual object tracking steps were omitted. Similarly, quantified fluorescence was averaged over all identified objects in the viewable field to create a single “blob” particle that was traced through time.

2.4 Acknowledgements

Portions of this chapter appear in part in Cookson S*; Ostroff N*; Pang WL*; Volfson D; Hasty J, “Monitoring dynamics of single-cell gene expression over multiple cell cycles”, *Molecular Systems Biology* doi:10.1038/msb4100032, Published online 22 November 2005. The dissertation author is a first author of this publication. Portions of this chapter appear in part in Pang WL; Bennett MR; Ostroff N; Hasty J, “Metabolic gene regulation in a dynamically changing environment”, *Nature*, which is in preparation for submission by Feb 2007. The dissertation author is the first author of this publication. Yeast variants WLPY001 and WLPY002 were developed with assistance from Jennifer Marciniak and utilized in an unpublished study.

3

Metabolic gene regulation in a dynamically changing environment

3.1 Introduction

Natural selection dictates that cells constantly adapt to dynamically changing environments in a context-dependent manner. For example, a sensitive response may be optimal when environmental changes are gradual, while a slow response may enable cells to save energy by ignoring rapid, transient environmental fluctuations. Gene-regulatory networks typically mediate the cellular response to perturbation[47, 9, 42], and an understanding of cellular adaptation will require experimental approaches aimed at subjecting cells to a dynamic environment that mimics their natural habitat[98, 69, 62, 60]. In this work, we monitored the response of *S. cerevisiae* metabolic gene regulation to periodic changes in the external carbon source by utilizing a microfluidic platform that allows precise, dynamic control over environmental conditions. We found that the metabolic system acts as

a low-pass filter that reliably responds to a slowly changing environment, while effectively ignoring fluctuations that are too fast for the cell to mount an efficient response. Using computational modeling calibrated with experimental data, we determined that frequency selection in the system is controlled by the interaction of coupled positive feedback networks governing the signal transduction of alternative carbon sources. More importantly, computational simulations suggested that the feedback loops may confer a robustness to environmental fluctuations on cells regardless of deficiencies in network components and we tested this prediction with an experimental comparison of two *S. cerevisiae* strains bearing significant phenotypic differences. In doing so, we found that the two exhibit the same filtering properties despite having markedly different induction and repression characteristics. These equivalent responses suggest that while the individual components of a complex regulatory network may differ when probed in a static batch-culture environment, the system as a whole has been optimized for a robust response to a dynamically changing environment. In the broader context of systems and quantitative biology, our findings establish an experimentally feasible framework for dynamically probing organisms in order to reveal the mechanisms that have evolved to mediate cellular responses to unpredictable environments.

In order to probe the response of individual cells to a fluctuating environment, we utilized the $T^2\mu C$ described in Chapter 2, a novel microfluidic platform capable of subjecting a population of cells to a continuously varying media supply (Figure 2.19). Briefly, the device was designed to generate a fluctuating media signal by dynamically combining two media reservoirs according to a user specified time-dependent function. Mixing of the source media streams was actuated by a fluidic input switch[36, 64] (Figure 2.19c) specially designed with an ideal, linear output response (Figure 2.19e). The resultant media

stream was then delivered downstream to a customizable monolayer growth chamber (Figure 2.19d), which for this study was constructed specifically for yeast cells, and allowed for long-term single-cell data acquisition[21]. The growth chamber received a continuous supply of fresh media via a series of “feeding” channels that coupled the chamber to the fluidic switch output. Placement of these feeding channels allowed for the rapid distribution of media throughout the growth chamber such that changes in the upstream supply would be transmitted practically instantaneously to the entire cellular population.

As a quantifiable reporter of the cellular response to environmental fluctuations, we fused genomic *GAL1* protein of *S. cerevisiae* to a yeast-optimized cyan fluorescent protein (yECFP). The enzymes for galactose utilization, including GAL1p, are among the most tightly regulated proteins in yeast. Because glucose requires much less energy to metabolize, cells will only consume galactose if glucose is not available [120]. Therefore, *S. cerevisiae* has evolved a highly complex regulatory network to ensure that the galactose enzymes will be strongly activated when they are needed, but tightly repressed if glucose is present in the environment. Because the network is well studied and involves regulatory motifs common to many higher organisms, galactose utilization has become a paradigm for the study of eukaryotic gene regulation. In order to build on the current understanding of its robust regulatory mechanisms, we employed our microfluidic platform to monitor the dynamics of network activation and repression in response to alternating galactose and glucose carbon sources.

3.2 Materials and methods

3.2.1 Microfluidic device fabrication

Microfluidic devices were fabricated using PDMS replica molding from silicon and photoresist masters as discussed in Chapter 2. Device designs were drawn using AutoCAD 2005 (Autodesk Inc.) and validated by a linear flow modeling program (see Appendix A) written in Matlab (The Mathworks). Photomasks were printed at 20,000 DPI onto photo-transparency film (Output City, Bandon OR) and used to pattern SU-8 (Microchem Corp.) onto clean silicon wafers. SU-8 photoresist was spun to appropriate depths and processed according to manufacturers specifications excluding development until the last layer was deposited and photo-patterned.

Primary fluidic channels (black and green in Fig. 2.19a) were fabricated to a depth of 10 μm . Grooves for the chaotic mixer were patterned at a total depth of 12 μm . Remaining geometric parameters for the mixing units were the same as those used by Stroock et al.[96]. and scaled accordingly to accommodate the designed channel depth. The chamber was fabricated to a depth of 4 μm to accommodate monolayer imaging of *S. cerevisiae*. Channels for on-chip temperature control (orange, Figure 2.19a) were patterned to a depth of at least 200 μm .

Poly-dimethylsiloxane (Sylgard 184, 2 part silicone encapsulant, Dow Corning) was mixed 1:10 (catalyst:base), degassed, and cast against the SU-8/silicon master mold, curing at 80°C for 1.5 hrs in a dry gravity oven (Fisher Scientific). Prior to casting the PDMS, chlorotrimethylsilane (Sigma) was vapor deposited for 2 min onto the master molds to aid in the release of cured PDMS monolith. Following release, the PDMS monolith was rinsed in 100% ethanol followed by deionized water. Fluidic ports for media and cell suspension were

punched using 20 ga luer stub adapters (McMaster-Carr). Ports for temperature control channels were punched using 16 ga stub adapters. These ports accommodated 23 ga and 18 ga, respectively, hypodermic steel tubing (Small Parts Inc) connection tips for fluidic lines leading to external fluid reservoirs. To remove debris from the port punching process, port holes were flushed with isopropanol and deionized water, and dried with high velocity stream of 0.2 μm filtered air. Any remaining debris on pattern surfaces was removed using a double application of office grade Scotch Tape (3M).

One day prior to use, the PDMS chips and 24 \times 40 mm, #1-1/2 coverslips were cleaned using a solvent rinse (acetone, isopropanol, methanol) and exposed to O₂ plasma for 1 min at 50 W in a Technics PEB-II plasma etcher. Treated surfaces were then brought into intimate contact to form an irreversible bond. Sealed chips were allowed to anneal overnight at 80°C to ensure maximum bond strength.[115]

All fluids used for microfluidic devices were 0.2 μm filtered to reduce channel occlusion by random debris and bacterial contamination. Similarly, all reservoirs and connecting lines were thoroughly cleaned by flushing with deionized water and isopropanol and autoclave sterilized prior to use. Devices were wetted using sterile water injected simultaneously at all waste outlet ports under 1–3 psi before attaching media and cell suspension reservoirs.

3.2.2 Cell preparation and culture

Transforming DNA was prepared by PCR amplification of linker + FP sequences + marker sequences found on the pKT0101a (Cerulean CFP) plasmid[93]. PCR primers were designed with approximately 40 bp of homologous sequences so that the FP sequences would be targeted to the carboxy-terminus of Gal1p, for pKT0090 and pKT0101a, respectively. To aid in transformation efficiency, 400 bp homology extension sequences that

flanked the insertion site and were anchored with the original 40 bp homologous sequence were cloned from genomic dna. Cells were transformed with the products from both fusion protein and genomic homology PCRs using heat shock and successful transformations were selected using histidine auxotrophy on synthetic dropout plates. Isolates that exhibited adequate fluorescence levels and growth on raffinose, galactose, and glucose, individually, were preserved as frozen stock for later use.

For subsequent experiments, frozen isolates were revived by streaking on to solid media plates. Plated cultures were at 30°C for 1–2 days and stored at 4°C for no longer than two weeks. One day prior to each experiment, an isolated colony was selected and grown in liquid medium containing 2% (w/v) raffinose at 30°C at 300 rpm overnight. On the experiment day, the overnight culture was diluted to an OD₆₀₀ 0.1 (if necessary) and grown to OD₆₀₀ 0.3–0.6 to ensure log phase growth.

3.2.3 Experimental conditions

Each experiment utilized three growth conditions: loading media, 2% raffinose; inducing media, 2% raffinose + 0.2% galactose; and repressing media, 2% raffinose + 0.2% galactose + 0.25% glucose. All media conditions were synthetic dropout based with appropriate supplements for auxotrophic selection. Prior to loading cells onto the microfluidic device, the cell suspension was synchronized by spiking the culture with 10 nM α_1 -mating factor and incubated for an additional hour. To maintain growth arrest during microfluidic device initialization, the loading media was also spiked with 10nM α_1 -mating factor. The repressing media also contained 0.01 mg/mL of sulforhodamine 101, a hydrophilic red fluorescent dye, to track the on-chip glucose concentration and location.

Microfluidic devices were primed with water followed by media using a specialized

pressurization system. Approximately 10–20 cells were loaded into the imaging field of the device. Once loaded, the device was set to perfuse only loading media through the growth chamber until the chip had reached thermal equilibrium at $30\pm 1^\circ\text{C}$ (approx. 30 min). The device flows were then changed so that only repressing media flowed through the chamber. The chip was imaged under these conditions for 10 min to establish a non-fluorescent (non-induced) baseline. Finally, the oscillatory drive was initiated and the chamber was imaged every 5 min for 24–48 hrs in red, and cyan fluorescence, and transmitted light.

3.2.4 Data analysis

Cells were segmented from the transmitted image sequence using a contrast enhancement algorithm. Cell occupied regions of each image were quantified for mean fluorescence in the cyan channel. Mean fluorescence of the background regions (non-cell occupied areas) in the red channel were measured as the glucose concentration state. Pixel area of the cell occupied regions were used to monitor on-chip growth. Population averaged trajectories of cyan fluorescence was normalized and detrended relative to the running trend average using an RLOWESS smoothing filter. Detrending utilized filter windows that varied based on the period of the oscillatory input. Detrended trajectory data was the result of subtracting data smoothed by a large window (containing the number of points equivalent to $2\times$ a period cycle) from data smoothed by a small window (points equivalent to half a period cycle). The value at each point of the detrended trajectory data was normalized by the corresponding value in the large window smoothed result.

No detrending was performed on glucose trajectory data (mean red fluorescence). Instead it was merely normalized relative to the range of intensity values and reported as 1-Normalized FL so the highest trajectory values would correspond to full galactose induction

conditions while the lowest trajectory values would correspond to full glucose repression. Data trajectories were truncated before the second cycle and after the maximum measured pixel area due to data processing artifacts and reduced segmentation efficiency, respectively. Amplitudes were calculated as the average vertical shift over all cycles from local maxima to local minima in the cyan fluorescence trajectory. Phase shifts were calculated as the average temporal delay between corresponding extrema in the reported “induction” signal and the detrended yellow and cyan fluorescence responses, scaled by a factor of $2\pi\omega$ where ω is the perturbation frequency in hr^{-1} .

3.3 Results and Discussion

3.3.1 Dynamic profiling

We setup our device to subject a population of yeast cells to sinusoidal glucose waves over a 0.2% (w/v) galactose background, varying the glucose concentration from 0.0% (no repression of *GAL1* transcription) to 0.25% (full repression; Figure 2.4). For each run we changed the frequency of the glucose signal, varying the period from 0.75–6 hrs, and imaged the population for a minimum of four full cycles (Figure 3.1). Time-lapse fluorescence imaging of the cell population in the growth chamber was used to calculate the amplitude ratio and phase shift of the cellular response relative to the glucose signal. The results show a maximum response frequency of about $5.58 \text{ rads hr}^{-1}$ (1.125 hr period). Above this frequency, the response trace was indistinguishable from a step response, whereas at the lower frequencies the temporal fluorescence trajectories clearly oscillated in response to the fluctuating input signal. In this sense, the galactose network appears to function as a low-pass filter, which enables the cells to ignore environmental fluctuations that are too

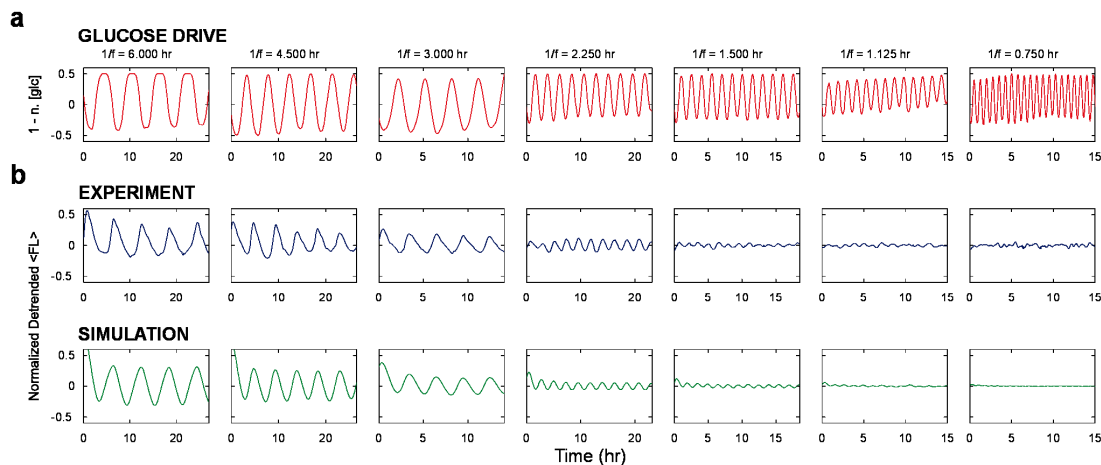


Figure 3.1: Simulated and experimental expression trajectories in response to sinusoidal perturbation at varying frequencies. **(a)** The induction drive signal used for both experimental and simulated responses. Glucose concentration is calculated from the mean fluorescence of a red tracer dye, normalized and subtracted from unity to represent the “induction” signal. Drive signal oscillation periods (left to right) are 6.0, 4.5, 3.0, 2.25, 1.5, 1.125, and 0.75 hrs. **(b)** Normalized and detrended fluorescence response for *S. cerevisiae* YPH499 cells under experimental (top) and simulated (bottom) conditions. Expression responses correspond to the oscillatory signals shown in (a).

frequent to mount an efficient response. The oscillation frequency at which this occurred was also interesting. The measured growth rate for these cells in the glucose based repression media was approximately 85–90 minutes, or 1.5 hours. The observed threshold fluctuation period was approximately 75% of this timescale. This may indicate that the cell cycle dynamics have ultimate control over global response timing, especially in the case of sugar metabolism.

3.3.2 Computational model

Since the sinusoidal driving of the galactose utilization network leads to complex cellular behavior, we used computational modeling to simulate the response and elucidate key aspects of the network architecture that give rise to the observed behavior. In particular, we were interested in how the interplay of the galactose and glucose utilization networks

gives rise to the low-pass filter characteristic and confers a robustness in cellular responses to carbon source fluctuations. Combined, the two pathways represent competing positive feedback networks that are coupled in two ways. First, detailed analysis of the galactose permease, Gal2p, has demonstrated that it not only actively transports galactose across the cell membrane, but glucose as well[73]. Second, internalized glucose activates *MIG1* expression, a transcription factor that strongly represses the galactose network. In the absence of glucose, Gal2p actively imports galactose from the extracellular environment, subsequently inducing its own expression via positive feedback. However, Gal2p is also capable of importing glucose, turning off *GAL2* transcription and turning on production of the machinery for glucose transport and metabolism, coupling the glucose response to galactose metabolism via negative feedback.

In order to simulate the effects of galactose activation and glucose repression on our experimental data, we incorporated the coupled feedback loops into a simple computational model of the cellular response to fluctuating carbon source (Fig. 3.2). Both the galactose and the glucose utilization networks of *S. cerevisiae* are complex gene networks incorporating many genes and proteins that tightly regulate the transport and metabolism of their respective sugar, and while the details of each differs greatly from the other, they both contain certain aspects that make them very similar. For instance, both networks are primarily designed to do equivalent tasks - i.e. metabolize their respective sugar when it is available. One design principle common to both networks is the positive feedback loop. For example, once a small amount of galactose has been internalized by the cell, it triggers a signalling cascade which ultimately increases the transcription rates of the genes responsible for its transport and metabolism. This, in turn, increases the internal galactose

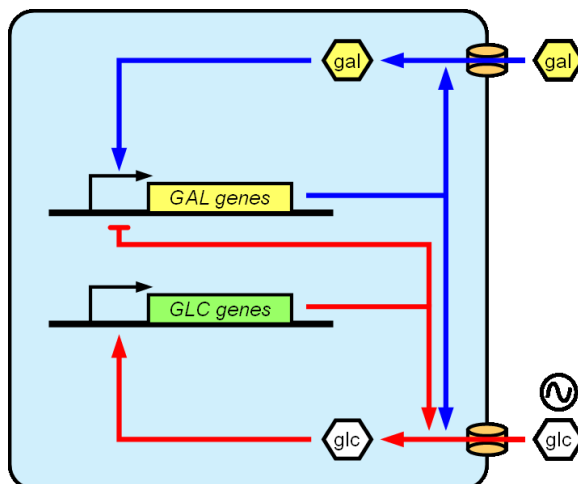


Figure 3.2: A network schematic of the computational model used. Reaction pathways governed by the galactose network are shown in blue, while those governed by the glucose network are drawn in red. Both networks contain positive feedback, in that their respective internalized sugars induce transcription of the transporters. Additionally, the two modules are coupled in two ways. First, the galactose permease can also transport glucose into the cell. Second, transcription factors of the glucose network repress the galactose module.

concentration, which further increases the gene activity.

Our computational model consisted of a system of ODEs that incorporated the main aspects of the coupled feedback loops described above. The model tracked the concentrations of the internal sugar, mRNA and proteins of each network. Instead of tracking every protein and reaction in the networks, our model is heuristic in that we assume that the activity of a network is representative of every protein produced by that network. In other words, we assume that the transcription factors of each network are proportional to one, representative protein. In this way, we were able to simplify our model to include just two proteins, representing the products of the two networks. We obtained parameters for both the galactose and glucose responses by (1) fitting steady state induction and repression curves acquired using flow cytometry and (2) modifying production and degradation rates to match the observed dynamic responses.

The coupling of the feedback modules was simulated as a two-part process. First,

the protein products of the glucose network globally repressed transcription of galactose network components. Second, the hexose transporter galactose network facilitated the internalization of glucose as well as galactose.

This led to the following governing equations,

$$\dot{m}_1 = \left\{ \frac{\alpha_1 + \epsilon_1 g_1^{\beta_1}}{\kappa_1^{\beta_1} + g_1^{\beta_1}} \right\} \left\{ \frac{\lambda^q}{\lambda^q + x_2^q} \right\} - \gamma_1 m_1 \quad (3.1)$$

$$\dot{x}_1 = \sigma_1 m_1 - \delta_1 x_1 \quad (3.2)$$

$$\dot{g}_1 = k_{tr} x_1 \left\{ \frac{[gal] - g_1}{k_{mtr} + [gal] + g_1 + \frac{a}{k_{mtr}} g_1 [gal]} \right\} \quad (3.3)$$

$$\dot{m}_2 = \left\{ \frac{\alpha_2 + \epsilon_2 g_2^{\beta_2}}{\kappa_2^{\beta_2} + g_2^{\beta_2}} \right\} - \gamma_2 m_2 \quad (3.4)$$

$$\dot{x}_2 = \sigma_2 m_2 - \delta_2 x_2 \quad (3.5)$$

$$\dot{g}_2 = k_{tr} (x_1 + x_2) \left\{ \frac{[glu] - g_2}{k_{mtr} + [glu] + g_2 + \frac{a}{k_{mtr}} g_2 [glu]} \right\}, \quad (3.6)$$

where $i = 1$ is the galactose network and $i = 2$ is the glucose network, and,

$m_i, x_i, g_i \Rightarrow$ mRNA, protein, and internal sugar concentrations, respectively

$\alpha_i, \epsilon_i \Rightarrow$ basal and activated transcription rates, respectively

$\sigma_i \Rightarrow$ global translation rate

$\gamma_i, \delta_i \Rightarrow$ degradation rates of the mRNA and proteins, respectively

$\beta_i, q \Rightarrow$ Hill coefficients for activated and repressed transcription, respectively

$\kappa_i, \lambda \Rightarrow$ concentrations for half-maximal induction and repression, respectively

$k_{tr}, k_{mtr}, a \Rightarrow$ constants describing the facilitated diffusion process

$[gal], [glu] \Rightarrow$ external concentrations of galactose and glucose

which was easily solved deterministically using built-in ODE solver routines in Matlab (The Mathworks).

We obtained model parameters for both the galactose and glucose responses by fitting steady state induction and repression curves acquired using flow cytometry and modifying production and degradation rates to match the observed step responses (Figure 3.3). The result is a model that accurately reproduces the dynamic response of a population of cells to sinusoidal repression over a large range of frequencies (Figure 3.1b).

3.3.3 Computational and experimental analysis of response robustness

Exploration with the model led to an interesting hypothesis regarding the need for such a tightly controlled coupling of the glucose and galactose networks: given this interplay, slight deficiencies in one of the networks might not hinder a cell's ability to adapt and thrive in a changing environment. In other words, a deficiency in one network would have the reverse effect in the other, leveling out the net response from the cell. The yeast strain used to collect this data, YPH499, is known to have a slight deficiency in the galactose utilization network, which causes it to require more galactose than "normal" to induce production of the galactose enzymes[104]. This deficiency is clear when comparing steady state induction data to that of another strain, K699, which is considered to have a more "wild-type" response to induction. Our flow cytometry population data (Fig. 2.3) demonstrates that YPH499 cells require about ten times more galactose to reach full induction than do K699 cells. Yet, despite this difference in steady state induction sensitivity, our model predicts that this deficiency need not translate into a less robust response to a fluctuating environment. Moreover, overcoming this insensitivity is achieved by merely changing the time it takes the glucose network to achieve full repression. When we changed the timing

of the glucose network in “wildtype” cellular simulations to be roughly four times faster than in YPH499, we found that this compensated for differences in steady state induction sensitivity suggesting that the complex pairing of the glucose and galactose networks confers robustness to the global response even when faced with seemingly detrimental network deficiencies.

To test this hypothesis, we turned our focus to the aforementioned K699 strain for experimental validation. We repeated the microfluidic runs at each frequency, this time using the K699 strain with an equivalent GAL1-yECFP fusion. Again, we chose parameters for the glucose and galactose responses based on fits to the steady state induction and repression population data (Figs. 2.3 & 2.4). As predicted, the amplitude responses of the two strains were strikingly similar (Figure 3.4), especially considering the significant difference in their galactose sensitivity.

To further validate our theory, we tested the glucose repression response times of each strain by loading them into a microfluidic device and subjecting them to a 10 hr induction pulse of galactose followed by a 10 hr repression pulse of glucose. Cellular fluorescence corresponding to Gal1p-CFP expression was measured and plotted versus time (Fig. 3.3 left). Where these trajectories peaked was determined as the onset of the glucose repression response. To aid in peak detection, the raw fluorescence trajectories were smoothed using a 5 point 1D averaging filter. The time value of the peak point for each trajectory was then subtracted from the time value when glucose was introduced to determine the response delay time for each strain (Fig. 3.3 right). YPH499 cells had a delay time of 0.583 hrs while K699 cells had a delay of 0.167 hrs, a 3.5 fold increase in response speed.

The quantitative similarity between the computational model and experimental

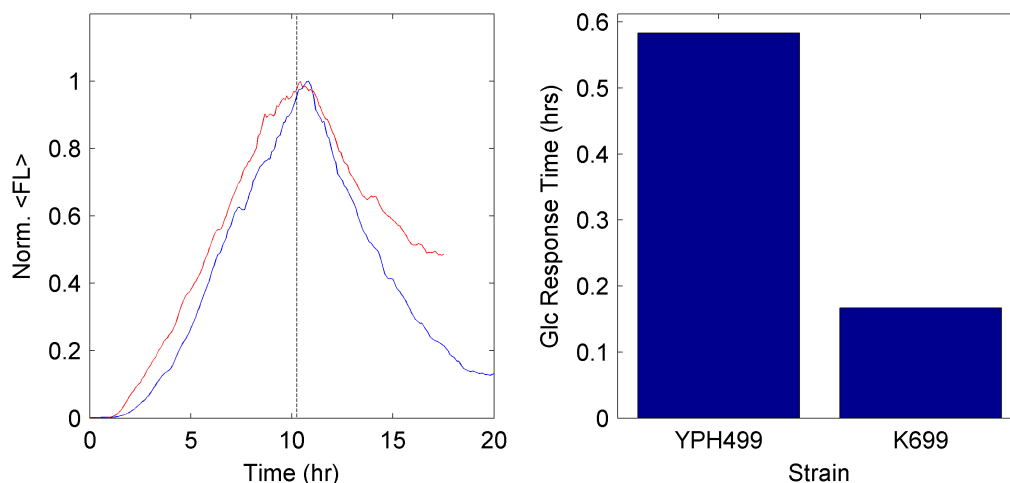


Figure 3.3: **(left)** Fluorescence trajectories of Gal1p-CFP expression in *S. cerevisiae* YPH499 (blue) and K699 (red) cells under a induction/repression double pulse experiment. The vertical black line represents the time at which the media state was switched from induction (galactose) to repression (glucose). **(right)** Glucose response delay times for *S. cerevisiae* YPH499 and K699 cells.

dynamic profiling results was truly impressive (Fig. 3.4), confirming both the existence of an informational filter as well as specific kinetic differences between the test strains. In addition to the amplitude and phase lag profiles, our model was able to predict several other experimentally observed differences between the strains. First, the average fluorescence was observed to be much higher in the K699 strain than in YPH499, a fact that suggests wild-type cells produce more GAL1p. Numerical simulations of our model corroborate this hypothesis and further suggest that it can explain the growth rate differences observed between the two strains during the trials. During the periodic stimulation, the K699 cells grew significantly faster than YPH499, with average on-chip doubling times of 6 hrs and 8 hrs, respectively. Our model suggests that while the amount of glucose metabolized by K699 is lower than that of YPH499, the increased level of GAL1p allows K699 to metabolize so much more galactose than YPH499 such that the total amount of both sugars metabolized

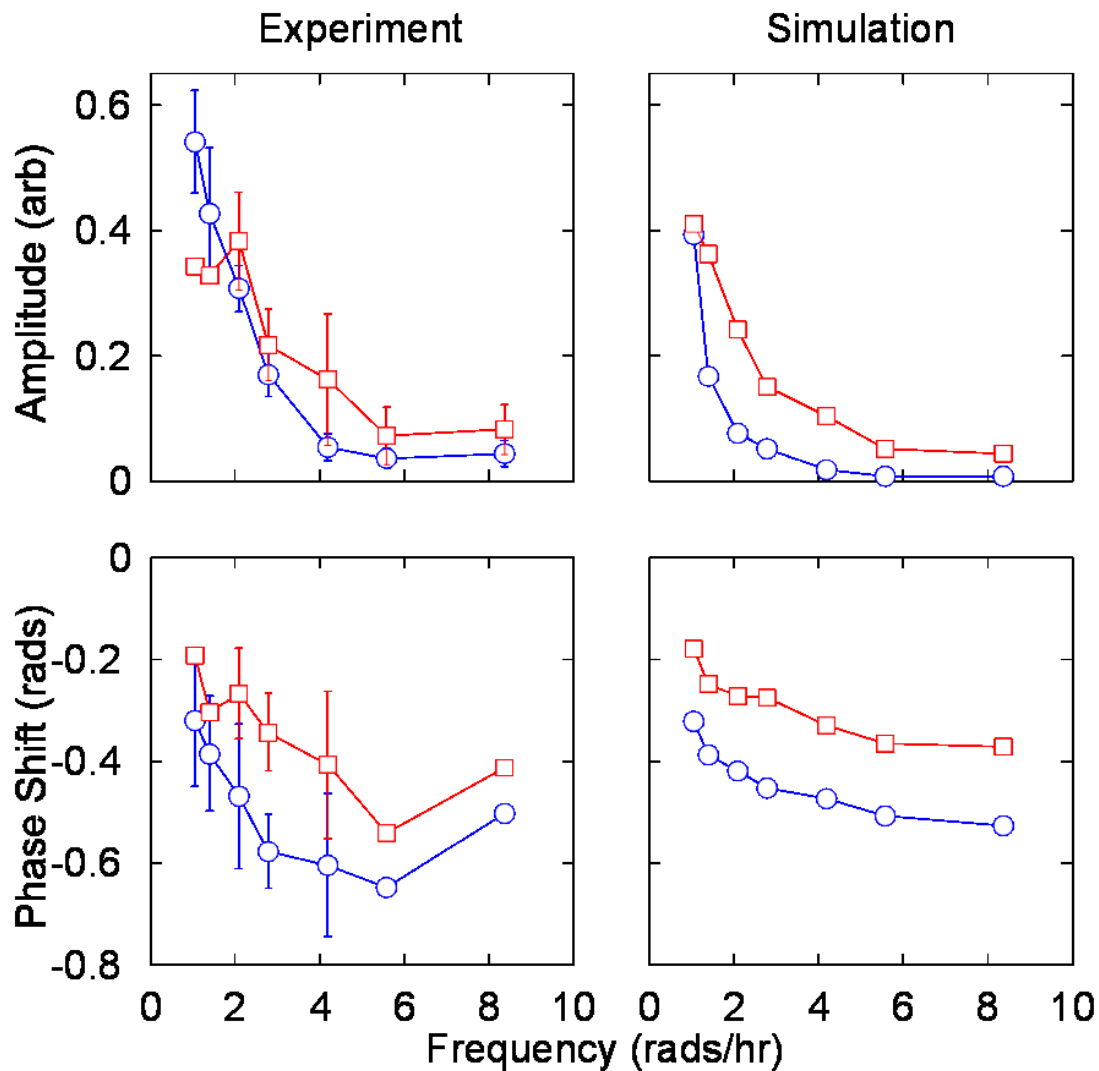


Figure 3.4: Frequency response profiles generated from experimental and computer simulated data. YPH499 data is shown as blue lines with circles. K699 data is shown as red lines with squares. Error bars for experimental data are calculated as standard error of the mean. Error bars for the first two K699 data points (corresponding to the lowest two frequencies) are omitted due to small sample population. Error bars for the last two data points (corresponding to the highest two frequencies) are omitted because fluctuations in gene expression responses at these frequencies were indistinguishable from noise.

is much greater in K699.

3.4 Conclusions

We have demonstrated the utility of a novel approach to studying biological systems, which combines computational analysis and dynamic expression data in a fluctuating environment to reveal defining network properties. Our results highlight the importance of studying organisms in more natural, dynamic environments, and its role in developing a deeper understanding of the complex networks that have evolved to ensure survival in non-static conditions. In addition, our model suggests an important interplay of the galactose and glucose utilization networks that optimizes cellular viability in such environment and there is increasing interest in further probing each network by investigating how other regulatory components contribute to the cellular response. Moreover, given the flexibility of our platform, focus is now placed on analyzing network responses to other types of fluctuating signals should they uncover additional network properties previously masked by steady-state analysis methods. Thus, using our approach, it is now possible to probe the networks of many model organisms, obtaining insight into the dynamic behavior of biological systems in their native habitats, uncovering previously unknown, yet information rich diversity of cellular regulatory phenomena.

3.5 Acknowledgements

The text of this chapter is a reprint of Pang WL; Bennett MR; Ostroff N; Hasty J, “Metabolic gene regulation in a dynamically changing environment”, *Nature*, which is in preparation for submission by Feb 2007.

4

Summary and Future Directions

4.1 Review

In this work I built foundational tools for complementing computational studies of single-cell dynamics. Toward this end, I developed and tested several microfluidic platforms capable of imaging isolated cellular populations under varying environmental conditions with high temporal resolution. Although not specifically used for the focus of this work, much of the technology survives and is being applied to other studies conducted by other members of my research group.

Most importantly, I developed a fluidic platform that is the first of many to replicate the dynamic environments that are commonplace throughout the natural world. Utilizing this unique platform, I discovered that the galactose utilization system, when perturbed by oscillations in external glucose concentration, functions as a rudimentary information filter whose response bandwidth may be controlled by the temporal constancy of the cell cycle. In addition, by applying a common engineering practice to the analysis of gene regulation, my colleagues and I have revealed that the coupling between the primary metabolic path-

ways in yeast, glucose and galactose metabolism, run far deeper than the simple boolean logic commonly associated with catabolite repression. Instead, the two pathways serve to complement each other in a dynamic environment, accounting for any kinetic deficiencies, and optimizing the viability of the cellular system as a whole.

4.2 Significance

Continued advancement in systems and quantitative biology relies on the development of experimental technologies that both complement and validate computational models. Many prior studies up until now have either utilized static (steady-state) measures of gene expression, or relied upon snap-shots of similar, but nonidentical populations of cells. In a similar thread, the existing literature tends to focus only on simple dynamic inputs and responses to those inputs, when in fact biological systems may experience wildly complex and dynamic stimuli in their natural environments. In order to proceed, the ability to track time evolved gene expression in single-cells in response to dynamic perturbation is paramount. The work performed here signifies a major step towards this.

4.3 Future directions

While the T² μ C and the data collected with *S. cerevisiae* is a major advancement towards the quantitative analysis of biological systems, several questions remain unanswered. For instance, the galactose system of *S. cerevisiae* was perturbed with a purely sinusoidal waveform of glucose. Discussions with colleagues of mine indicate that similar, if not more, data could be extracted from high frequency pulses to the system. Moreover, it would be equivalently interesting to study cellular responses to pure environmental “noise”,

or hybrid waveforms with both sinusoidal components followed by random fluctuations. An even more interesting test would be to see if prolonged exposure to oscillatory nutrient perturbation could entrain the cell cycle or even bias a population towards a subgroup that bares a specific phenotype.

While the above proposed work serves to further test the responsiveness of the galactose system as a whole, and to what perturbation it is best adapted to respond to, one could equally head in the opposite direction and analyze components of the network in isolation. This would allow researchers to gain better insight into how simple dynamic processes lead to globally complex ones, helping to improve predictive models of the system. Lastly, other reasonably well known pathways such as glucose repression, oxidative stress, alpha mating, filamentation, and more could be studied in the like, to the point where *S. cerevisiae* would become the first organism to be fully characterized from a dynamic standpoint.

Finally, a noted limitation of the current platform is that it is only capable of observing one microcolony of cells under one specific perturbation. To this end, it is necessary to parallelize the platform to allow for either the simultaneous screening of genetically different microcolonies, parallel perturbation probes, or both. While the XLC and dynamic gradient devices I presented can be easily adapted to account for the former two scenarios, complete parallelization would require surmounting current limitations in microfluidic device fabrication and flow control methods.

4.4 Closing

As a whole, this work integrates mathematical and engineering theory, as well as advanced microtechnology, to address scientific questions. The tools developed here will serve to collect previously unreachable answers and help to ask new and more insightful questions. Lastly, the knowledge contributed by this work and related studies will significantly augment our knowledge of biological processes and how they result in a complex and dynamic behavior.

A

MOCA: Microfluidic Open Circuit Analyzer

A.1 Basic description, requirements, and concepts

The name is pronounced like the chocolate flavored espresso drink, but it is not as tasty. MOCA stands for Microfluidic Open Circuit Analyzer and is a Matlab routine for simulating flows in microfluidic channel networks that do not possess digital control elements such as on-chip valves.

The core of the code is the solution of the linear system of equations that represent the on-chip pressures and flowrates on an interconnected network of microchannels. This only constitutes about 50 lines of code. The remaining amount is devoted to making the algorithm general enough to use for nearly any microfluidic network and producing human readable textual and graphical displays of simulation results.

A.1.1 Requirements

The primary routine uses only core Matlab functions and should be able to run on Matlab version 6 and higher. The graphical display uses some gui elements available through Matlab's GUIDE utility and may only run on platform versions 7 and higher. At the time of this writing the user interface requires that network model specification be entered by hand. A graphical interface for defining network connectivity and geometry is planned and will be made available once complete.

A.1.2 Concepts

When developing “analog” microfluidic devices, a microfluidic designer often reaches the following solution for the Navier-Stokes equations of fluid motion,

$$\Delta P = Q \cdot \left(\frac{8\mu L}{\pi r^4} \right) \quad (\text{A.1})$$

where r and L are the radius and length of a cylindrical pipe, respectively. This is remarkably similar to Ohm's law for analog electrical circuits,

$$V = I \cdot R \quad (\text{A.2})$$

where pressure, P , flow rate, Q , and the lumped material and geometric constants fall into the roles of potential, current, and resistance, respectively. A similar analysis done on a channel with a rectangular cross-section[10] yields,

$$\Delta P = Q \cdot \left(\frac{12\mu L}{wd^3} \right) \cdot \alpha \quad (\text{A.3})$$

where,

$$\alpha = \left[1 - a \left(\frac{192}{\pi^5} \sum_{n=1,3,5}^{\infty} \frac{1}{n^5} \tanh \left(\frac{n\pi}{2a} \right) \right) \right]^{-1} \quad (\text{A.4})$$

and,

$$a = \frac{d}{w} \quad (\text{A.5})$$

Note for small values of a , i.e. $w \gg d$, α tends toward unity and the fluidic resistance through such a rectangular channel is simply,

$$R_f = \frac{12\mu L}{wd^3} \quad (\text{A.6})$$

While Eqns. A.3 to A.6 are adequate to describe flow through a single microfluidic channel, the majority of microfluidic devices are comprised of complex networks of interconnected channels. Solving the system of equations derived from these complex networks by hand can be very time consuming endeavor. This is where the MOCA aids the designer.

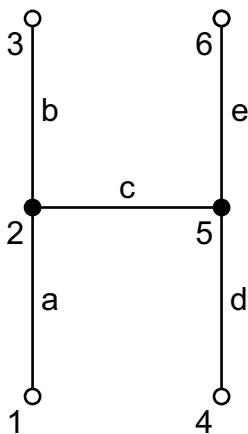


Figure A.1: Node/Segment schematic of a fluidic h-cross

For example, the fluidic network shown in figure A.1, an h-cross, is the simplest network that best demonstrates the MOCA's usefulness. Performing a microfluidic open circuit analysis as described above gives the following equation for the internal nodes 2 and

5,

$$\begin{pmatrix} P_2 \\ P_5 \end{pmatrix} = \begin{pmatrix} G_{abc} & -G_c \\ -G_c & G_{cde} \end{pmatrix}^{-1} \begin{pmatrix} G_a & G_b & 0 & 0 \\ 0 & 0 & G_d & G_e \end{pmatrix} \begin{pmatrix} P_1 \\ P_3 \\ P_4 \\ P_6 \end{pmatrix} \quad (\text{A.7})$$

or,

$$P_{internal} = G_{internal}^{-1} G_{external} P_{external} \quad (\text{A.8})$$

where,

$$G_i = \frac{1}{R_i} \quad (\text{A.9})$$

and,

$$G_{abc} = G_a + G_b + G_c \quad (\text{A.10})$$

Once the internal node pressures $P_{internal}$ are known, all the segment flows in the device are determined using the following equation.

$$\begin{pmatrix} Q_a \\ Q_b \\ Q_c \\ Q_d \\ Q_e \end{pmatrix} = \begin{pmatrix} Ga & 0 & 0 & 0 & 0 \\ 0 & Gb & 0 & 0 & 0 \\ 0 & 0 & Gc & 0 & 0 \\ 0 & 0 & 0 & Gd & 0 \\ 0 & 0 & 0 & 0 & Ge \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \end{pmatrix} \quad (\text{A.11})$$

or,

$$Q_i = G_{ij} C_{jk} P_k \quad (\text{A.12})$$

where G_{ij} is a matrix with the segment conductance values as elements and C_{jk} is a constant matrix that specifies segment connections between the nodes P_k .

Note there are several major assumptions made by the MOCA. First, all channels simulated have simple cross sectional geometry; rectangular with smooth walls. Complex topologies, such as curved channels or channels with corrugated walls cannot be simulated and approximations for the changes in flow characteristics must be made by the user. Second, materials compressibility is assumed negligible. Under typical operating pressures (0–20 inH₂O), PDMS, a silicone rubber used for rapid prototyping of devices, behaves as if it were as solid as glass. Lastly, the interaction of other transport forces, such as forced convection by thermal transfer, are not accounted for.

A.2 Simulating a device

To simulate flow for any microfluidic system, the user must supply the MOCA the following,

- pressure values at external node points,
- node linkages that form flow segments,
- and geometric specifications for each segment.

This is done by specifying two Matlab cell arrays, **press** which defines the pressure nodes, and **conn** which specifies both node/segment connectivity, and segment geometry. For the h-cross system described above this would look like,

```
press = {
    [0 0], 15, 'inh2o', 'node 1';
    [0 1], nan, '', 'node 2';
    [0 2], 10, 'inh2o', 'node 3';
5    [1 0], 15, 'inh2o', 'node 4';
    [1 1], nan, '', 'node 5';
    [1 2], 12, 'inh2o', 'node 6';
};
```

```

10 conn = {
    [1 2], 1.000, 0.075, 0.025, 'a', 'segment a';
    [2 3], 1.000, 0.075, 0.025, 'b', 'segment b';
    [2 5], 3.000, 0.075, 0.025, 'c', 'segment c';
    [4 5], 1.000, 0.075, 0.025, 'd', 'segment d';
15    [5 6], 1.000, 0.075, 0.025, 'e', 'segment e';
};

```

The rows of the `press` cell array represent unique nodes within the fluidic system, and are listed in the order the nodes are assigned – e.g. the first row is for node id 1. Each row contains five columns: a vector of xy coordinates (for display purposes), the pressure value at the node, the pressure units, and a brief description of the node point – e.g. “cell inlet port”.

The rows of the `conn` cell array represent unique segments within the fluidic system. Each row contains six columns. The first column specifies which two nodes the segment connects. Note the numbers in this vector represent node ids and not spatial coordinates. The next three columns define the geometry of the segment, length, width, and depth, respectively, in units of millimeters. The fifth column is a segment id used for graphical display. The last column is used for a brief description of the segment – e.g. “switch output channel”.

Simulating flow is done by typing `moca(conn, press)` at the Matlab command prompt. This will generate the following output,

```

>> moca(conn, press)
segment      ul s^-1  nl s^-1  mm s^-1  tau (s)  l (mm)  w (mm)  d (mm)
a:segment a   0.0456  45.6329  24.3376  0.0411   1.000   0.075   0.025
b:segment b   0.0504  50.4364  26.8994  0.0372   1.000   0.075   0.025
5 c:segment c  -0.0048 -4.8035  -2.5618  1.1710   3.000   0.075   0.025
d:segment d   0.0312  31.2225  16.6520  0.0601   1.000   0.075   0.025
e:segment e   0.0264  26.4191  14.0902  0.0710   1.000   0.075   0.025
node          Pa      inh2o    psi
1              3735.9900  15.0000  0.5417
10 2           3144.4583  12.6250  0.4559
3              2490.6600  10.0000  0.3611
4              3735.9900  15.0000  0.5417

```

5	3331.2578	13.3750	0.4830
6	2988.7920	12.0000	0.4334

along with the following figure,

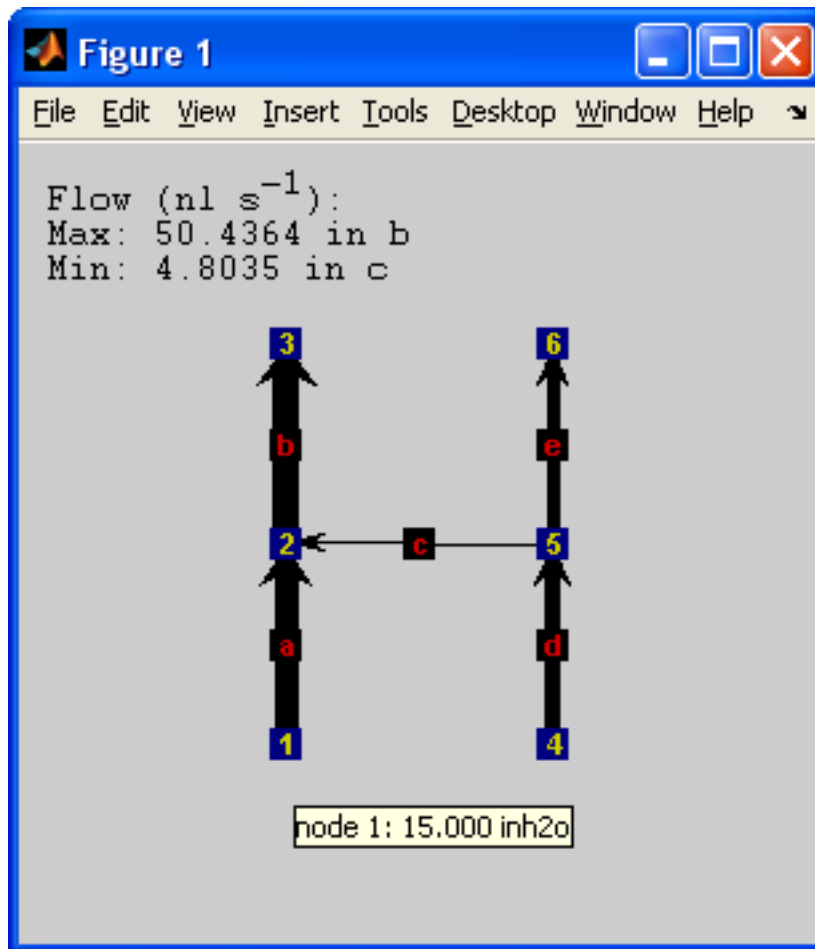


Figure A.2: Graphical output of a MOCA simulation of an h-cross microfluidic system. The tooltip is displayed when hovering the mouse pointer over the node label. Similar tooltips are available for the segment labels.

Note, the reason that segment “c” has a negative flowrate in the textual output is due to it’s direction in the graphical display. The segment was defined with a positive direction as leaving node 2 and entering node 5. However, with the pressures specified, simulated flow runs in the reverse direction, hence the negative value. If the segment were specified going the other way, e.g. from node 5 to node 2, the flow rate value would be

positive.

In the graphical display of the simulation results, the magnitudes and directions of flow are indicated by the width and direction of arrows connecting node points, respectively. For convenience the maximum and minimum flow rate values are displayed in the upper left hand corner of the figure window. In addition, when the mouse cursor is hovered over a node or segment id label, information about that element is displayed in a tooltip bubble. This currently is restricted to the element label provided in the last columns of `press` and `conn` and the node pressure, or segment flowrate.

A.3 Common approximations

A.3.1 Channels in parallel

At times it may be necessary to simplify the conceptual fluidic network to reduce display complexity. This generally happens when there are multiple channels in parallel such as the feeding channels for the $T^2\mu C$, or in linear gradient mixers. Reducing parallel channels is done in much the same way parallel resistances are reduced in electric circuits.

For instance, the flow through a parallel bed of channels is equivalent to the sum of flow through all its constituent channels,

$$Q_{tot} = \sum_{i=1} Q_i = \sum_{i=1} \frac{P_i}{R_i} \quad (\text{A.13})$$

Assuming that the pressure drop over all the channels is constant,

$$Q_{tot} = P \sum_{i=1} \frac{1}{R_i} = P \sum_{i=1} G_i \quad (\text{A.14})$$

where for rectangular channels,

$$\sum_{i=1} G_i = G_{eff} = \frac{w_{eff} d_{eff}^3}{12\mu L_{eff}} \quad (\text{A.15})$$

In the case of identical parallel channels, it is easier to keep d_{eff} and L_{eff} the same as the original depth and length. Thus,

$$G_{eff} = \frac{d^3}{12\mu L} \sum_{i=1} w_i \quad (\text{A.16})$$

where the value of $\sum_{i=1} w_i$ is the width specified for the “effective” channel segment in conn.

A.3.2 Channels in series

Multiple channels with varying geometry placed in series can also be reduced to a single “effective” channel. In this case the pressure drop over the entire series is,

$$P_{tot} = \sum_{i=1} Q_i R_i \quad (\text{A.17})$$

As before, flow through the effective unit should be conserved such that,

$$P_{tot} = Q_{tot} \sum_{i=1} R_i = Q_{tot} \sum_{i=1} \frac{12\mu L_i}{w_i d_i^3} \quad (\text{A.18})$$

where,

$$R_{eff} = \sum_{i=1} \frac{12\mu L_i}{w_i d_i^3} \quad (\text{A.19})$$

Under these circumstances it is easier to convert the geometry of all channels to be equivalent in width and depth, thus varying only in length to maintain the channel resistance. Thus,

$$R_{eff} = \frac{12\mu}{wd^3} \sum_{i=1} L_i \quad (\text{A.20})$$

A.3.3 Valved channels

Although not specifically designed to handle “digital” control of fluid flow, the net effects of having integrated flow valves in microfluidic devices can be simulated. This

can be done by either setting the channel depth to zero, or the channel length to infinity (Inf). Under these circumstances the MOCA will display a dashed line for the channel in the graphical output indicating no flow through the segment.

A.4 MOCA models for devices used in this work

A.4.1 $T\mu C$

Model parameters and states

```

% loading conditions
pressld = { [0 2], 25.00, 'inh2o', 'cells';
            [1 2], nan,   '',      '';
            [2 1], nan,   '',      '';
5           [2 2], nan,   '',      '';
            [3 0], 12.50, 'inh2o', 'waste';
            [3 2], nan,   '',      '';
            [3 4], 17.50, 'inh2o', 'media';
            };
10 % operating conditions
pressop = { [0 2], 17.50, 'inh2o', 'cells';
            [1 2], nan,   '',      '';
            [2 1], nan,   '',      '';
            [2 2], nan,   '',      '';
15           [3 0], 12.50, 'inh2o', 'waste';
            [3 2], nan,   '',      '';
            [3 4], 25.00, 'inh2o', 'media';
            };
conn = { [1 2], 9.500, 0.150, 0.012, 'a', 'cell load';
20         [2 3], 0.320, 0.120, 0.004, 'b', 'diode entrance';
         [2 4], 0.165, 0.044, 0.012, 'c', '';
         [3 4], 0.176, 0.040, 0.004, 'd', 'diode exit';
         [4 6], 9.500, 0.040, 0.012, 'e', '';
25         [5 6], 3.500, 0.040, 0.012, 'f', 'waste';
         [6 7], 3.500, 0.040, 0.012, 'g', 'media';
            };

```

Simulation Output

```

>> moca(conn, pressld)
segment      ul s^-1  nl s^-1  mm s^-1  tau (s)  l (mm)  w (mm)  d (mm)
a:cell load  0.0009  0.8581  0.4767  19.9269  9.500   0.150   0.012
b:diode entrance 0.0000  0.0190  0.0395  8.0977  0.320   0.120   0.004
5 c:         0.0008  0.8392  1.5893  0.1038  0.165   0.044   0.012
d:diode exit 0.0000  0.0190  0.1186  1.4846  0.176   0.040   0.004
e:         0.0009  0.8581  1.7878  5.3138  9.500   0.040   0.012

```

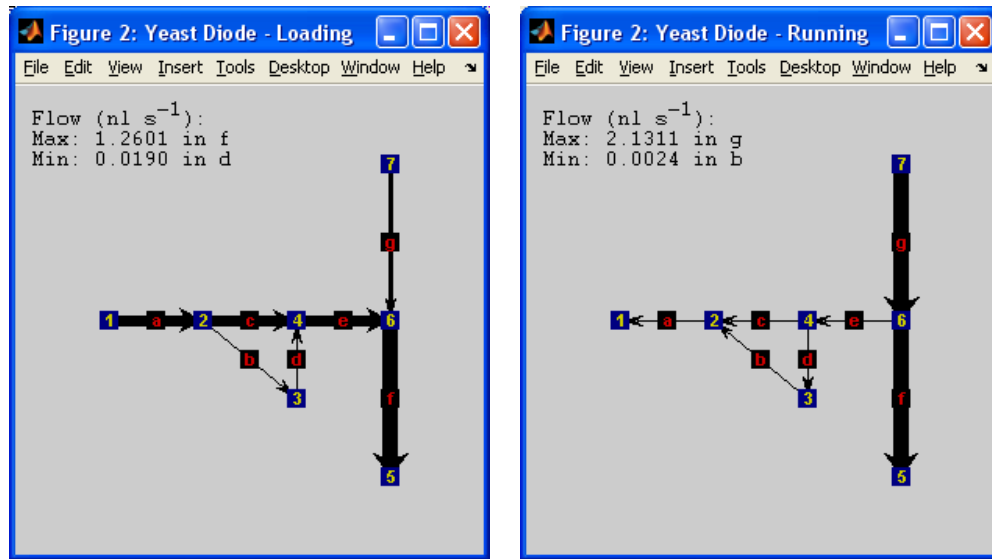


Figure A.3: Graphical results for MOCA simulation of the $T_{\mu}C$ microfluidic device

```

f:waste          -0.0013  -1.2601  -2.6251   1.3333   3.500   0.040   0.012
g:media          -0.0004  -0.4019  -0.8373   4.1799   3.500   0.040   0.012
10 node          Pa      inh2o      psi
 1              6226.6501  25.0000   0.9029
 2              5829.1885  23.4042   0.8452
 3              5819.5008  23.3653   0.8438
 4              5802.7992  23.2982   0.8414
15 5              3113.3250  12.5000   0.4514
 6              4057.4937  16.2908   0.5883
 7              4358.6550  17.5000   0.6320

20 >> moca(conn, pressop)
segment          ul s^-1  nl s^-1  mm s^-1  tau (s)  l (mm)  w (mm)  d (mm)
a:cell load      -0.0001  -0.1073  -0.0596  159.4150  9.500   0.150   0.012
b:diode entrance -0.0000  -0.0024  -0.0049   64.7816   0.320   0.120   0.004
c:               -0.0001  -0.1049  -0.1987   0.8305   0.165   0.044   0.012
25 d:diode exit  -0.0000  -0.0024  -0.0148  11.8766   0.176   0.040   0.004
e:               -0.0001  -0.1073  -0.2235  42.5107   9.500   0.040   0.012
f:waste          -0.0020  -2.0238  -4.2163   0.8301   3.500   0.040   0.012
g:media          -0.0021  -2.1311  -4.4398   0.7883   3.500   0.040   0.012
node            Pa      inh2o      psi
30 1              4358.6550  17.5000   0.6320
 2              4408.3377  17.6995   0.6392
 3              4409.5487  17.7043   0.6394
 4              4411.6364  17.7127   0.6397
 5              3113.3250  12.5000   0.4514
35 6              4629.7996  18.5886   0.6713
 7              6226.6501  25.0000   0.9029

```

A.4.2 T²μC

Model parameters and states

```

% simulation of d0041 (ttmc with optimized daw switch)
press_base = { ...
    [0 2], 20.00, 'inh2o', 'input 1';
    [0 4], 10.00, 'inh2o', 'waste s';
5    [0 5], 8.00, 'inh2o', 'waste c';
    [0 6], nan, '', '';
    [0 7], 3.00, 'inh2o', 'cells';
    [1 2], nan, '', '';
    [2 0], 19.50, 'inh2o', 'shunt';
10   [2 1], nan, '', '';
    [2 2], nan, '', 'switch interface';
    [2 3], nan, '', 'switch output';
    [2 4], nan, '', 'filter barrier channel side';
    [2 5], nan, '', 'filter barrier chamber side';
15   [2 6], nan, '', 'chamber entrance';
    [3 2], nan, '', '';
    [4 2], 20.00, 'inh2o', 'input 2';
    [4 5], 8.00, 'inh2o', 'waste m';
    [4 6], nan, '', '';
20   [4 7], 14.50, 'inh2o', 'media';
    };

% defines differences between states. mostly just changes in
% applied pressures to external nodes.
25 press_load = press_base;
    press_load( 5, 2:3) = { 6, 'psi'};
    press_load(18, 2:3) = { 6, 'psi'};

    press_op_1 = press_base;
30 press_op_1( 1, 2:3) = {25, 'inh2o'};

    press_op_2 = press_base;
    press_op_2(15, 2:3) = {25, 'inh2o'};

35
% geometry as l, w, d, in mm
conn = { ...
    [ 3 4], 1.500, 0.100, 0.015, 'a', 'waste c';
    [ 4 5], 1.500, 0.100, 0.015, 'b', 'cells';
40   [ 1 6], 3.000, 0.075, 0.015, 'c', 'input 1';
    [ 2 11], 5.215, 0.150, 0.015, 'd', 'waste s';
    [ 4 13], 7.875, 0.100, 0.015, 'e', '';
    [ 6 8], 0.500, 0.050, 0.015, 'f', 'bypass input 1';
    [ 6 9], 0.150, 0.075, 0.015, 'g', '';
45   [ 7 8], 2.000, 0.150, 0.015, 'h', 'shunt';
    [ 8 9], 0.150, 0.120, 0.015, 'i', 'guide channels to shunt';
    % parallel approximation.
    [ 9 10], 0.200, 0.120, 0.015, 'j', 'guide channels to chamber';
    % parallel approximation.
50   [10 11], 13.500, 0.150, 0.015, 'k', 'switch output';
    % accounts for mixing banks not explicitly modeled

```



```

[11 12], 0.050, 0.210, 0.001, 'l', 'feeding/filter channels';
    % parallel approximation.
[12 13], 0.600, 0.100, 0.004, 'm', 'chamber';
55     % only long arm modeled
[ 8 14], 0.500, 0.050, 0.015, 'n', 'bypass input 2';
[ 9 14], 0.150, 0.075, 0.015, 'o', '';
[14 15], 3.000, 0.075, 0.015, 'p', 'input 2';
[13 17], 8.510, 0.100, 0.015, 'q', '';
60 [16 17], 1.500, 0.100, 0.015, 'r', 'waste m';
    [17 18], 1.500, 0.100, 0.015, 's', 'loading media';
};

```

Simulation Output

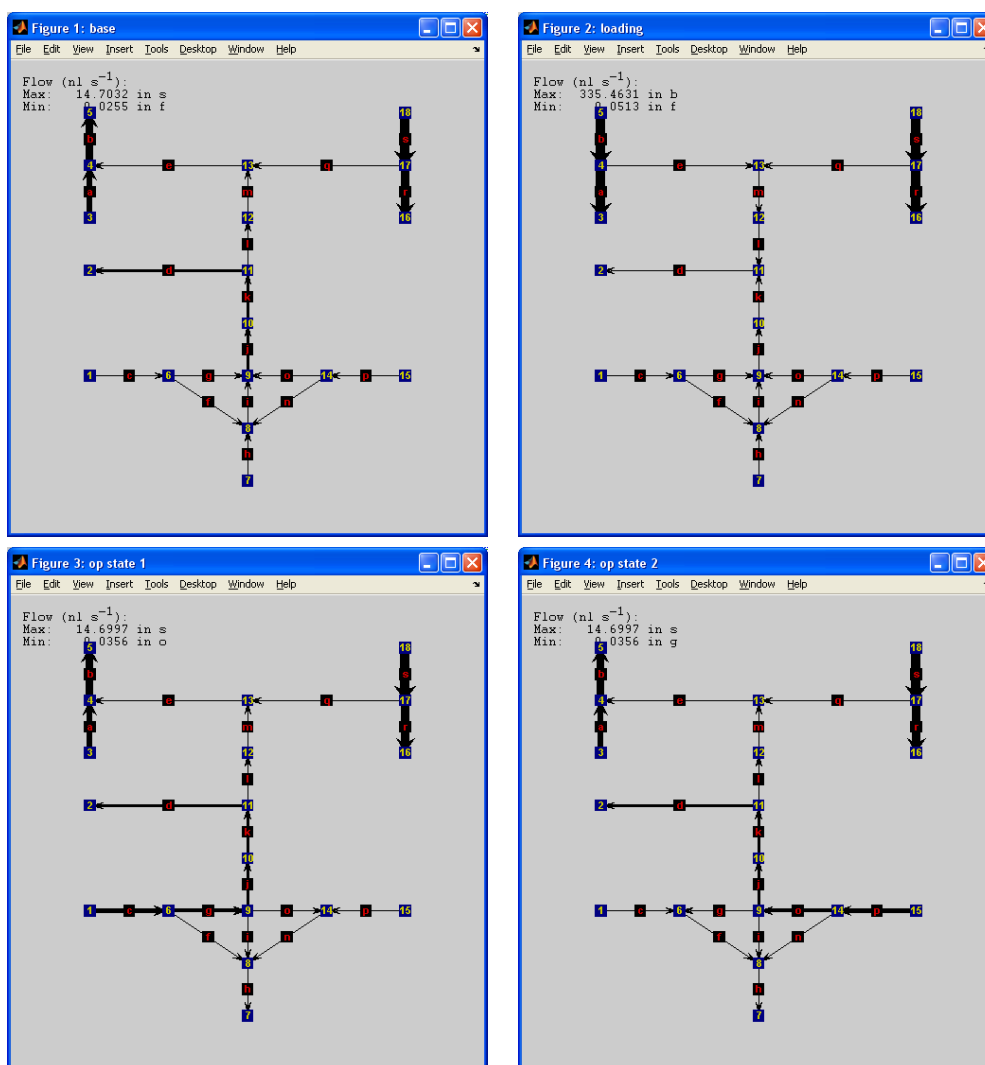


Figure A.4: Graphical results for MOCA simulation of the T²μC microfluidic device

```

>> moca(conn, press_base, 'figsize', [500 500], 'nodesize', 0.025)
segment      ul s^-1  nl s^-1  mm s^-1  tau (s)  l (mm)  w (mm)  d (mm)
a:waste c    0.0095   9.4883   6.3255   0.2371   1.500   0.100   0.015
b:cells      0.0117  11.6542   7.7695   0.1931   1.500   0.100   0.015
5 c:input 1  0.0014   1.3813   1.2278   2.4434   3.000   0.075   0.015
d:waste s    -0.0045  -4.5213  -2.0095   2.5952   5.215   0.150   0.015
e:           -0.0022  -2.1659  -1.4439   5.4539   7.875   0.100   0.015
f:bypass input 1 0.0000   0.0255   0.0340  14.6895   0.500   0.050   0.015
g:           0.0014   1.3557   1.2051   0.1245   0.150   0.075   0.015
10 h:shunt    0.0020   2.0034   0.8904   2.2461   2.000   0.150   0.015
i:guide channels 0.0021   2.0545   1.1414   0.1314   0.150   0.120   0.015
   to shunt
j:guide channels 0.0048   4.7660   2.6478   0.0755   0.200   0.120   0.015
   to chamber
15 k:switch output 0.0048   4.7660   2.1182   6.3733  13.500   0.150   0.015
l:feeding/filter 0.0002   0.2447   1.1653   0.0429   0.050   0.210   0.001
   channels
m:chamber    0.0002   0.2447   0.6118   0.9807   0.600   0.100   0.004
n:bypass input 2 -0.0000  -0.0255  -0.0340  14.6895   0.500   0.050   0.015
20 o:         -0.0014  -1.3557  -1.2051   0.1245   0.150   0.075   0.015
p:input 2    -0.0014  -1.3813  -1.2278   2.4434   3.000   0.075   0.015
q:           -0.0019  -1.9212  -1.2808   6.6444   8.510   0.100   0.015
r:waste m    -0.0128 -12.7820 -8.5214   0.1760   1.500   0.100   0.015
s:loading media -0.0147 -14.7032 -9.8021   0.1530   1.500   0.100   0.015
25 node      Pa      inh2o      psi
1            4981.3200  20.0000    0.7223
2            2490.6600  10.0000    0.3611
3            1992.5280   8.0000    0.2889
4            1433.6503   5.7561    0.2079
30 5          747.1980   3.0000    0.1083
6            4756.5394  19.0975    0.6897
7            4856.7870  19.5000    0.7042
8            4755.4201  19.0930    0.6895
9            4745.5081  19.0532    0.6881
35 10         4714.8500  18.9301    0.6837
11           3087.1491  12.3949    0.4476
12           2385.8462   9.5792    0.3459
13           2103.4169   8.4452    0.3050
14           4756.5394  19.0975    0.6897
40 15         4981.3200  20.0000    0.7223
16           1992.5280   8.0000    0.2889
17           2745.4124  11.0228    0.3981
18           3611.4570  14.5000    0.5237

45 >> moca(conn, press_load, 'figsize', [500 500], 'nodesize', 0.025)
segment      ul s^-1  nl s^-1  mm s^-1  tau (s)  l (mm)  w (mm)  d (mm)
a:waste c    -0.3332 -333.2230 -222.1487  0.0068   1.500   0.100   0.015
b:cells      -0.3355 -335.4631 -223.6421  0.0067   1.500   0.100   0.015
c:input 1    0.0012   1.1518   1.0238   2.9301   3.000   0.075   0.015
50 d:waste s  -0.0079  -7.9179  -3.5191   1.4819   5.215   0.150   0.015
e:           0.0022   2.2401   1.4934   5.2733   7.875   0.100   0.015
f:bypass input 1 0.0001   0.0513   0.0683   7.3156   0.500   0.050   0.015
g:           0.0011   1.1006   0.9783   0.1533   0.150   0.075   0.015
h:shunt      0.0013   1.2878   0.5723   3.4944   2.000   0.150   0.015
55 i:guide channels 0.0014   1.3903   0.7724   0.1942   0.150   0.120   0.015
   to shunt

```

```

j:guide channels 0.0036 3.5914 1.9952 0.1002 0.200 0.120 0.015
  to chamber
k:switch output 0.0036 3.5914 1.5962 8.4577 13.500 0.150 0.015
60 l:feeding/filter -0.0043 -4.3265 -20.6025 0.0024 0.050 0.210 0.001
  channels
m:chamber -0.0043 -4.3265 -10.8163 0.0555 0.600 0.100 0.004
n:bypass input 2 -0.0001 -0.0513 -0.0683 7.3156 0.500 0.050 0.015
o: -0.0011 -1.1006 -0.9783 0.1533 0.150 0.075 0.015
65 p:input 2 -0.0012 -1.1518 -1.0238 2.9301 3.000 0.075 0.015
q: -0.0021 -2.0865 -1.3910 6.1180 8.510 0.100 0.015
r:waste m -0.3333 -333.2998 -222.1999 0.0068 1.500 0.100 0.015
s:loading media -0.3354 -335.3863 -223.5909 0.0067 1.500 0.100 0.015
node Pa inh2o psi
70 1 4981.3200 20.0000 0.7223
  2 2490.6600 10.0000 0.3611
  3 1992.5280 8.0000 0.2889
  4 21619.9471 86.8041 3.1349
  5 41379.3103 166.1379 6.0000
75 6 4793.8788 19.2474 0.6951
  7 4856.7870 19.5000 0.7042
  8 4791.6313 19.2384 0.6948
  9 4784.9238 19.2115 0.6938
  10 4761.8214 19.1187 0.6905
80 11 3535.2704 14.1941 0.5126
  12 15934.0091 63.9750 2.3104
  13 20927.2398 84.0229 3.0344
  14 4793.8788 19.2474 0.6951
  15 4981.3200 20.0000 0.7223
85 16 1992.5280 8.0000 0.2889
  17 21624.4711 86.8223 3.1355
  18 41379.3103 166.1379 6.0000

>> moca(conn, press_op_1, 'figsize', [500 500], 'nodesize', 0.025)
90 segment ul s^-1 nl s^-1 mm s^-1 tau (s) l (mm) w (mm) d (mm)
a:waste c 0.0095 9.4845 6.3230 0.2372 1.500 0.100 0.015
b:cells 0.0117 11.6580 7.7720 0.1930 1.500 0.100 0.015
c:input 1 0.0074 7.3973 6.5753 0.4562 3.000 0.075 0.015
d:waste s -0.0050 -4.9842 -2.2152 2.3542 5.215 0.150 0.015
95 e: -0.0022 -2.1734 -1.4490 5.4350 7.875 0.100 0.015
f:bypass input 1 0.0012 1.2394 1.6526 0.3026 0.500 0.050 0.015
g: 0.0062 6.1578 5.4736 0.0274 0.150 0.075 0.015
h:shunt -0.0022 -2.2082 -0.9814 2.0379 2.000 0.150 0.015
i:guide channels -0.0009 -0.8787 -0.4881 0.3073 0.150 0.120 0.015
100 to shunt
j:guide channels 0.0052 5.2435 2.9131 0.0687 0.200 0.120 0.015
  to chamber
k:switch output 0.0052 5.2435 2.3305 5.7929 13.500 0.150 0.015
l:feeding/filter 0.0003 0.2593 1.2347 0.0405 0.050 0.210 0.001
105 channels
m:chamber 0.0003 0.2593 0.6482 0.9256 0.600 0.100 0.004
n:bypass input 2 -0.0001 -0.0901 -0.1201 4.1634 0.500 0.050 0.015
o: 0.0000 0.0356 0.0317 4.7353 0.150 0.075 0.015
p:input 2 -0.0001 -0.0544 -0.0484 62.0015 3.000 0.075 0.015
110 q: -0.0019 -1.9141 -1.2761 6.6688 8.510 0.100 0.015
r:waste m -0.0128 -12.7855 -8.5237 0.1760 1.500 0.100 0.015
s:loading media -0.0147 -14.6997 -9.7998 0.1531 1.500 0.100 0.015
node Pa inh2o psi

```

```

1          6226.6501  25.0000  0.9029
115 2          2490.6600  10.0000  0.3611
3          1992.5280  8.0000  0.2889
4          1433.8723  5.7570  0.2079
5           747.1980  3.0000  0.1083
6          5022.8563  20.1668  0.7283
120 7          4856.7870  19.5000  0.7042
8          4968.5126  19.9486  0.7204
9          4972.7517  19.9656  0.7210
10         4939.0217  19.8302  0.7162
11         3148.2297  12.6401  0.4565
125 12         2405.2026  9.6569  0.3488
13         2105.9700  8.4555  0.3054
14         4972.4617  19.9644  0.7210
15         4981.3200  20.0000  0.7223
16         1992.5280  8.0000  0.2889
130 17         2745.6192  11.0237  0.3981
18         3611.4570  14.5000  0.5237

>> moca(conn, press_op_2, 'figsize', [500 500], 'nodesize', 0.025)
segment      ul s^-1  nl s^-1  mm s^-1  tau (s)  l (mm)  w (mm)  d (mm)
135 a:waste c      0.0095  9.4845  6.3230  0.2372  1.500  0.100  0.015
    b:cells        0.0117  11.6580  7.7720  0.1930  1.500  0.100  0.015
    c:input 1      0.0001  0.0544  0.0484  62.0015  3.000  0.075  0.015
    d:waste s     -0.0050  -4.9842  -2.2152  2.3542  5.215  0.150  0.015
    e:             -0.0022  -2.1734  -1.4490  5.4350  7.875  0.100  0.015
140 f:bypass input 1 0.0001  0.0901  0.1201  4.1634  0.500  0.050  0.015
    g:            -0.0000  -0.0356  -0.0317  4.7353  0.150  0.075  0.015
    h:shunt       -0.0022  -2.2082  -0.9814  2.0379  2.000  0.150  0.015
    i:guide channels -0.0009  -0.8787  -0.4881  0.3073  0.150  0.120  0.015
        to shunt
145 j:guide channels 0.0052  5.2435  2.9131  0.0687  0.200  0.120  0.015
        to chamber
    k:switch output 0.0052  5.2435  2.3305  5.7929  13.500  0.150  0.015
    l:feeding/filter 0.0003  0.2593  1.2347  0.0405  0.050  0.210  0.001
        channels
150 m:chamber      0.0003  0.2593  0.6482  0.9256  0.600  0.100  0.004
    n:bypass input 2 -0.0012  -1.2394  -1.6526  0.3026  0.500  0.050  0.015
    o:            -0.0062  -6.1578  -5.4736  0.0274  0.150  0.075  0.015
    p:input 2     -0.0074  -7.3973  -6.5753  0.4562  3.000  0.075  0.015
    q:            -0.0019  -1.9141  -1.2761  6.6688  8.510  0.100  0.015
155 r:waste m     -0.0128  -12.7855  -8.5237  0.1760  1.500  0.100  0.015
    s:loading media -0.0147  -14.6997  -9.7998  0.1531  1.500  0.100  0.015
        node
            Pa      inh2o      psi
1          4981.3200  20.0000  0.7223
2          2490.6600  10.0000  0.3611
160 3          1992.5280  8.0000  0.2889
4          1433.8723  5.7570  0.2079
5           747.1980  3.0000  0.1083
6          4972.4617  19.9644  0.7210
7          4856.7870  19.5000  0.7042
165 8          4968.5126  19.9486  0.7204
9          4972.7517  19.9656  0.7210
10         4939.0217  19.8302  0.7162
11         3148.2297  12.6401  0.4565
12         2405.2026  9.6569  0.3488
170 13         2105.9700  8.4555  0.3054

```

14	5022.8563	20.1668	0.7283
15	6226.6501	25.0000	0.9029
16	1992.5280	8.0000	0.2889
17	2745.6192	11.0237	0.3981
175 18	3611.4570	14.5000	0.5237

A.5 Complete source code for moca.m

```

function [varargout] = moca(conn, press, varargin)
% Microfluidic Open Circuit Analyzer
%   moca(conn, press, ...)
%   P           = moca(conn, press, ...)
5 %   [P, Q]     = moca(conn, press, ...)
%
%   moca(conn, press) runs the routine with the segment connection data in
%   conn and the node pressure data in press.  Each input is a cell array
%   with each row consisting of data for each segment and node for conn and
10 %   press respectively.
%
%   When called without output arguments, moca will default to both verbose
%   (data output to the command window) and graphical results display.
%   These can be overridden by setting the optional parameters 'verbose'
15 %   and 'graphic' to false.
%
%   When called with output arguments, moca returns P, a vector of node
%   pressures in Pa and/or Q a vector of segment flows in  $\mu\text{L s}^{-1}$ .
%
20 %   Connectivity Data:
%   conn = {[<node id 1> <node id 2>], length, width, depth, id, label}
%         node id # corresponds to the row number of the specific node in
%         press.
%
25 %         channel dimensions (length, width, depth) should be in millimeters
%
%         id should be a single character, e.g. a, b, c ..., and is used to
%         label the segment in the graphical display.
%
30 %         label is a brief description of the channel segment, e.g. 'media
%         line'.
%
%   All elements of conn must be defined (e.g. not Inf or NaN).
%
35 %   Node Pressure Data:
%   press = {coord (x, y), pressure value, pressure unit, label}
%           coord (x, y) is a 1 x 2 integer vector containing the cartesian
%           coordinates of each node.  This is used for graphical results
%           display.
40 %
%           pressure value must be set for all 'external' nodes, e.g. nodes
%           having only one immediate neighbor for the algorithm to work
%           appropriately.  For 'internal' nodes, e.g. nodes where the pressure
%           is not explicitly known, the value should be set to NaN.  The
45 %           algorithm will solve for these values.

```

```

%
%     pressure unit can be 'inh2o', 'psi', or 'pa'.  internally, all
%     pressures are converted to 'pa'.  For internal nodes, set the
%     pressure unit to an empty string ('');
50 %
%     label is a brief description of the node, e.g. waste port.
%
% Optional Parameters:
% mu             default: 0.001
55 %             fluid viscosity value with units of Pa-s.
%
% verbose       default: depends on number of output arguments
%               set to true if textual simulation results are desired
%
% graphic       default: depends on number of output arguments
%               set to true if graphical simulation results are desired.
%               If false, the paramters specified by 'figsize' and
%               'nodesize' are ignored.
60 %
%
% figsize       default: [300 300]
%               figure dimensions in pixels for graphical results display.
%               this parameter is ignored if 'graphic' is set to false.
65 %
%
% nodesize      default: 0.04
%               size of graphical labels (n x n) in normalized figure
70 %               units.  Ignored if 'graphic' is set to false.
%
% Tip:
% if graphical display is specified, hover the mouse pointer over node
75 % and segment labels to view the pressure or flow rate for that
% particular node or segment, respectively.
%
% See moca_examples.m for conn and press specification and general usage
% examples.
80
% History:
% 2006/03/30 WLP:   Created.
% 2006/03/31 WLP:   Made the generation of the external and internal
%                   conductance matrices (GE, GI) more generic --
85 %                   supports external nodes with multiple connected
%                   segments.
%                   Graphic display now displays flows at or near zero
%                   as a dotted line without an arrow head.
% 2006/04/12 WLP:   Included the full analytical solution for channel
90 %                   resistance from Beebe DJ, Mensing GA, and Walker
%                   GM. Annu Rev Biomed Eng Vol 4, p261 (2002).
%                   Accounts for large aspect ratio channels (e.g. when
%                   the hieght is >> than the width)
% 2006/07/19 WLP:   Added/Fixed segment display to allow for
95 %                   multicharacter ids.
% 2006/09/02 WLP:   Manually wrapped long lines of code so that they
%                   are more readable.  Updated some of the comments.
%
% =====
100 % NO USER EDITING BEYOND THIS POINT!

opts    = getoptx(varargin);

```

```

mu      = parseopts('mu', opts, 0.001); % Pa s, fluid viscosity
iFigSz  = parseopts('figsize', opts, [300 300]);
105 iFigWd = iFigSz(1);
iFigHt  = iFigSz(2);
nNodeSz = parseopts('nodesize', opts, 0.04);

bVerbose= parseopts('verbose', opts, nargout == 0);
110 bGraphic= parseopts('graphic', opts, nargout == 0);
bDebug  = parseopts('debug', opts, false);
bExpt   = parseopts('experimental', opts, false);

% determine node connectivity and node type. External nodes are nodes that
115 % have a constrained (user set) pressure and may have any number of
% connections. Internal nodes should be undefined and set to 'nan'.

S = reshape([conn{:, 1}]', 2, [])'; % node-segment matrix
n = unique(S); % node vector
120
ne = sort(find(~isnan([press{:, 2}])))'; % external nodes
ni = sort(find( isnan([press{:, 2}])))'; % internal nodes

if bDebug, ni, ne, end;
125
% determine node segment connections and neighbor nodes
% nconn = {node, connected segments, neighbor nodes}
nconn = {};
NN = fliplr(S);
130 for i = 1:length(n),
    idx = find(S == n(i));
    [rw cl] = ind2sub(size(S), idx);
    nconn(i, :) = {n(i), rw, NN(idx)};
end;
135 if bDebug, nconn, end;

% channel geometry
l = [conn{:, 2}]; % mm, length
w = [conn{:, 3}]; % mm, width
140 d = [conn{:, 4}]; % mm, depth
s = [conn(:, 5)]; % segment id, use cell indexing since ids can be strings

% determine which is smaller between w and d since this affects aspect
% ratio and resistance calculations
145 wold = w; dold = d;
idx = find(w < d);
tmp = w;
w(idx) = d(idx);
d(idx) = tmp(idx);
150
a = d./w; % channel aspect ratio

R = 12*l*mu./(w.*d.^3); % Pa-s mm^-3, channel resistances
R = R.*rfact(a); % full analytical solution for resistance
155
G = 1./R; % mm^3 (Pa-s)^-1 channel conductances

% restore the w and d vector's to user specified values to avoid confusion
% in the verbose display

```

```

160 w = wold; d = dold;

    % build the external and internal conductance matrices.
    % GI will always be diagonally symmetric and square nxn where n is the
    % number of internal nodes. GE will be nxm where m is the number of
165 % external nodes driving the system.

    % initialize the conductance matrices
    GI = zeros(length(ni), length(ni));
    GE = zeros(length(ni), length(ne));
170
    % loop through the list of nodes
    for i = 1:size(press, 1),
        % process only the internal nodes (these are what we need to solve
        % for). External nodes that are only connected to other external nodes
175 % do not affect the solution.

        % if the node is internal (pressure unspecified)
        if isnan(press{i, 2}),
            irow = find(ni == i);
180             GI(irow, irow) = -sum(G(nconn{i, 2}));

            % loop over neighbors
            for j = 1:length(nconn{i, 3}),
                nn = nconn{i, 3}(j); % neighbor node id
185                 ns = nconn{i, 2}(j); % neighbor seg id

                % add entries to the GI matrix for each internal neighbor
                if ismember(nn, ni),
                    icol = find(ni == nn);
190                     GI(irow, icol) = G(ns);
                    GI(icol, irow) = GI(irow, icol); % symmetry of GI matrix
                end;

                % add entries to the GE matrix for each external neighbor
195                 if ismember(nn, ne),
                    ecol = find(ne == nn);
                    GE(irow, ecol) = -G(ns);
                end;

            end; % neighbor loop
        end; % internal node?
    end; % node loop

    P = [press{:, 2}]';
205
    % convert all pressures values to Pa
    for i = 1:size(press, 1),
        switch lower(press{i, 3}),
            case 'inh2o',
210                 P(i) = P(i)*1000/4.015;
            case 'psi',
                P(i) = P(i)*1000/0.145;
            case 'pa',
                % do nothing
215                 case '',
                    % assume internal node or units are already Pa

```



```

        % do nothing
        otherwise,
            error('Unrecognized pressure unit!');
220    end;
    end;

    if bDebug,
        GI, GE
225    printf('%s: %d x %d', 'GI', size(GI));
        printf('%s: %d x %d', 'PI', size(P(isnan(P))));
        printf('%s: %d x %d', 'GE', size(GE));
        printf('%s: %d x %d', 'PE', size(P(~isnan(P))));
    end;
230
    % solve for unknown or unset pressures
    P(isnan(P)) = inv(GI)*GE*P(~isnan(P));

    % determine segment flowrates
235 % build connectivity matrix
    C = zeros(length(G), length(P));
    for i = 1:size(conn, 1),
        C(i, conn{i, 1}(1)) = 1;
        C(i, conn{i, 1}(2)) = -1;
240 end;

    % calculate flows:
    Q = diag(G)*C*P; % ul s-1

245
    % =====
    % RESULTS AND GRAPHIC DISPLAY
    if nargout > 0,
        if nargout == 1,
250            varargout{1} = P;
        elseif nargout == 2,
            varargout{1} = P;
            varargout{2} = Q;
        end;
255 end;

    if bVerbose,
        printf('%-22s%10s%10s%10s%10s%10s%10s', ...
            'segment', 'ul s-1', 'nl s-1', 'mm s-1', ...
260            'tau (s)', 'l (mm)', 'w (mm)', 'd (mm)');
        for i = 1:size(conn, 1),
            printf('%s:%-20s%10.4f%10.4f%10.4f%10.4f%10.3f%10.3f%10.3f', ...
                conn{i, 5:6}, Q(i), Q(i)*1000, Q(i)/w(i)/d(i), ...
                abs(l(i)/(Q(i)/w(i)/d(i))), l(i), w(i), d(i));
265        end;
        printf('%-22s%10s%10s%10s', 'node', 'Pa', 'inh2o', 'psi');
        for i = 1:length(P),
            printf('%-22s%10.4f%10.4f%10.4f', ...
                num2str(i), P(i), P(i)*4.015/1000, P(i)*0.145/1000);
270        end;
    end;

    if bGraphic,

```

```

% display the flows in a figure
275 hfig = figure;
    pos = get(gcf, 'position');
    set(hfig, 'position', [25 50 iFigWd iFigHt])

nodes = [[1:size(press, 1)]' reshape([press{:, 1}]', 2, [])'];
280 % segs = reshape([conn{:,2}], 4, [])';

% build the segment coord matrix from the node-segment matrix and the
% node coords supplied in press.
segs = zeros(size(S, 1), 4);
285 for i = 1:size(press, 1),
    [rw cl] = find(S == i);
    for j = 1:length(rw),
        offset = (cl(j)-1)*2;
        segs(rw(j), 1+offset:2+offset) = press{i, 1};
290    end;
end;

% draw the segment arrows, pointing in the direction of flow. positive
% is up and right, and is indicated by the order of the segment
295 % endpoint coordinates
x = [segs(:, 1) segs(:, 3)]; x = (x + 1)/(max(x(:)) + 2);
y = [segs(:, 2) segs(:, 4)]; y = (y + 1)/(max(y(:)) + 2);

% arrow width normalized by the max abs flow value
300 iMaxWidth = 8; % max point width for arrows
p = abs(Q); p = p/max(p)*iMaxWidth;
p(find(p < 0.5)) = 0.5;
for i = 1:length(p),
    ax = x(i, :);
305    ay = y(i, :);
    if Q(i) < 0,
        ax = fliplr(ax);
        ay = fliplr(ay);
    end;
310
    ahead = iMaxWidth;
    if p(i)*3 > iMaxWidth, ahead = p(i)*3; end;
    if abs(Q(i)*1000) > 0.0001,
        annotation(hfig, 'arrow', ax, ay, ...
315         'headwidth', ahead, ...
         'headlength', 12, ...
         'linewidth', p(i));
    else,
        annotation(hfig, 'line', ax, ay, ...
320         'linewidth', p(i), ...
         'linestyle', ':');
    end;

% display the segment label at the midpoint of the segment
325 % use a uicontrol instead of an annotation object since the tooltip
% property can be used to display the object information
if isempty(conn{i, 6}),
    lbl = 'unlabeled';
else,
330    lbl = conn{i, 6};

```

```

end;
uicontrol(hfig, 'style', 'text', ...
    'units', 'normalized', ...
    'position', [mean(ax)-nNodeSz/2, mean(ay)-nNodeSz/2, ...
335                                     nNodeSz, nNodeSz], ...
    'string', s(i), ...
    'fontsize', 8, ...
    'fontname', 'monotype', ...
    'fontweight', 'bold', ...
340    'foregroundcolor', [0.8 0 0], ...
    'backgroundcolor', 'k', ...
    'horizontalalignment', 'center', ...
    'tooltipstring', sprintf('%s: %.3f nL s-1', ...
        lbl, abs(Q(i))*1000) );
345 end;

% draw the node points
% convert abs coords to normalized figure units with at least one node
% spacing around the margin of the figure
350 x = nodes(:, 2); x = (x + 1)/(max(x) + 2);
y = nodes(:, 3); y = (y + 1)/(max(y) + 2);
wd = nNodeSz; ht = nNodeSz; % size of the nodes
for i = 1:size(nodes, 1);
    if isempty(press{i, 4}),
355        lbl = 'unlabeled';
    else,
        lbl = press{i, 4};
    end;
    uicontrol(hfig, 'style', 'text', ...
360        'units', 'normalized', ...
        'position', [x(i)-wd/2, y(i)-ht/2, wd, ht], ...
        'string', num2str(nodes(i, 1)), ...
        'fontsize', 8, ...
        'fontname', 'monotype', ...
365        'fontweight', 'bold', ...
        'foregroundcolor', [0.8 0.8 0], ...
        'backgroundcolor', [0 0 0.5], ...
        'horizontalalignment', 'center', ...
        'tooltipstring', sprintf('%s: %.3f inh2o', ...
370        lbl, P(i)*4.015/1000) );
end;

% display max flow and min flow
sFlows = sprintf(...
375    'Flow (nl s-1):\nMax:%10.4f in %s\nMin:%10.4f in %s', ...
        max(abs(Q))*1000, s{find(abs(Q) == max(abs(Q)), 1)}, ...
        min(abs(Q))*1000, s{find(abs(Q) == min(abs(Q)), 1)});
    annotation(hfig, 'textbox', [0.01, 0.51, 0.48, 0.48], ...
        'string', sFlows, ...
380        'fitheighttotext', 'on', ...
        'fontname', 'courier', ...
        'linestyle', 'none');
end;

385 % The nested function below calculates the multiplicative factor used in
% determining the fluidic resistance in a channel with rectangular cross
% section. See Beebe DJ, Mensing GA, and Walker GM. Annu Rev Biomed Eng

```

```
% Vol 4, p261 (2002) for more details.
function C = rfact(a, varargin)
390 if nargin == 2,
    nmax = abs(varargin{2});
    if ~mod(nmax, 2)
        if nmax > 2,
            nmax = nmax - 1;
395     else,
            nmax = nmax + 1;
        end;
    end;
else,
400     nmax = 99;
    end;
    n = [1:2:nmax];

    C = [];
405 for i = 1:length(a)
    C(i) = (1 - a(i)*192/pi^5*sum((1./n.^5).*tanh(n.*pi/2/a(i))))^-1;
end;
```

B

IMSQT: IMage Segmentor, Quantifier, Tracker

B.1 Overview

The work described in this dissertation required specialized multichannel imaging for data collection and quantification. In all cases, images were taken using transmitted light and one or more fluorescence channels. Because fluorescent reporter proteins were linked to dynamically expressed host proteins, and fluorescently traced expression stimuli was equally dynamic, fluorescent images could not be used for image segmentation. Instead, transmitted light images were used, and because the microscopy system lacked phase contrast optics, special contrast enhancement algorithms were required to separate object features from image backgrounds. In various test cases, image objects were also tracked through time using their object centroids. A crucially important requirement was the ability to manually validate and correct every major step in the data analysis to ensure the utmost accuracy of

the extracted data.

While commercial applications were available at the time of this writing with similar capabilities, they lacked the ability to be fully customized and automated to suit the needs of this work. In the end, it was more feasible to create a custom architecture for extracting data from long time-series image sets with multiple tracked objects and multiple quantification channels.

The IMSQT is actually a small suite of graphical subprograms written in Matlab. The main application window, accessible by typing `imsqt` at the Matlab command prompt, is the portal to each individual subprogram, as well as the location for defining analysis parameters and saving analysis sessions for future use. The other main programs within the suite include:

segedit A graphical segmentation editor. Displays algorithm generated object segmentation boundaries over the image(s) used for segmentation. Provides the ability to edit segmentation masks either manually or using morphological filters and binary mask operations.

trackedit A graphical object tracking validation tool. Displays segmented objects with tracking identification numbers. Used to determine and account for algorithm errors, which generally include mislabeled or lost objects.

trackview A command line tool for displaying quantified object trajectory data. Generates multipanel figures of the data if specified, as well as outputs requested figure data for subsequent analysis.

Internal to the application are three operating scripts that produce the desired segmentation, quantification, and tracked object data. These are:

`imsegauto` A segmentation controller.

`imsegcollect` An image/object data collector.

`imsegtrack` An object tracking routine.

B.2 Main Window

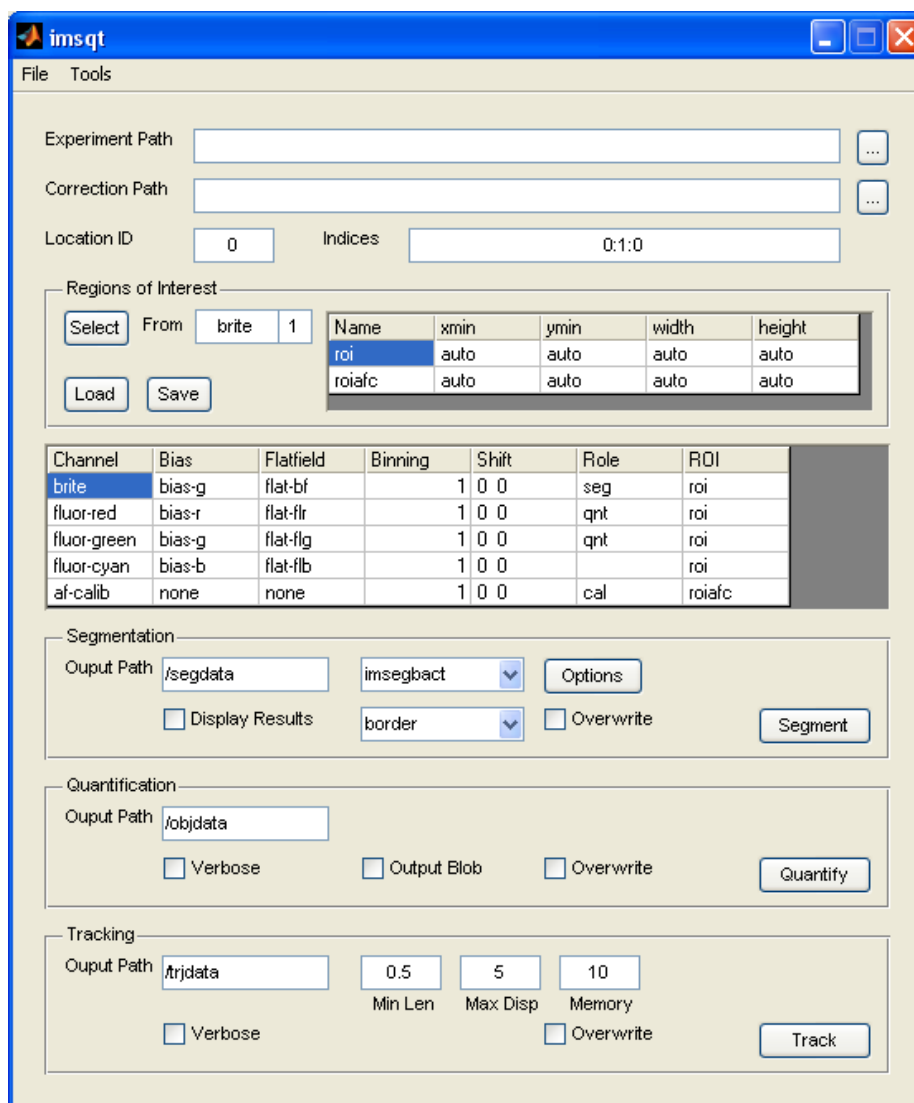


Figure B.1: The IMSQT main window

Prior to performing any image segmentation, quantification, or tracking details of the imaging session must be specified.

B.2.1 Experiment Path

The root directory for the imaging session. This is where subdirectories containing individual imaging channels.

B.2.2 Correction Path

The root directory for correction image channels. This may be either an independent directory or a subdirectory within the experiment path. The imaging channels housed in this directory must correspond to the correction image channels specified in the channel table.

B.2.3 Location ID

This is the location identifier for the image set being analyzed. This must be a numeric value, and for fixed location image sets this is 0. For imaging sessions that have multiple locations, this number must be incremented by hand. Subsequent analysis steps (segmentation, quantification, and tracking) are designed to operate on a single location at a time.

B.2.4 Indices

A numeric listing, that specifies the image indices to include in the analysis. Note, the program recognizes the last 4 characters in an image file name (prior to the extension) as the image index. For instance, an image ending in `..._0150.tiff` would be interpreted

as having index 150. Internally, the indices listing is interpreted as a Matlab vector. Thus, indices listings “1, 2, 3, 4, 5” and “1:1:5” are equivalent. Mixed notation may also be used — e.g. “1:1:5, 20:1:50” skips over images 6–19.

B.2.5 Regions of Interest

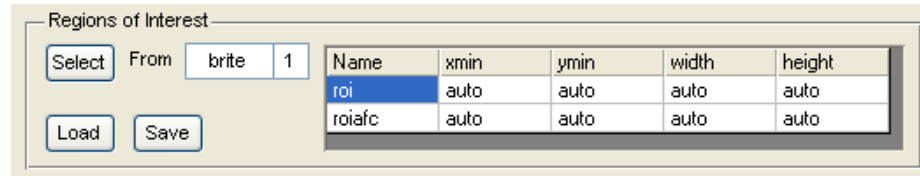


Figure B.2: Regions of interest definition panel in the IMSQT main window.

These are user specified areas within an image that the subsequent processing and analysis will be performed on. These regions are global to the image set and are used on a per channel basis, that is, each channel must specify the region of interest (ROI) that is desired.

An ROI is defined as a rectangular area within the image using coordinates xmin, ymin, width, and height. Coordinates must be positive and > 0 . The default values for these coordinates are “auto”, which specifies that the entire image will be used. The user may graphically select a region of interest by specifying the channel and image ID (next to “From”) to be used and clicking the “Select” button. ROI sets may be saved by clicking the “Save” button. Similarly, a previously saved ROI set may be retrieved using the “Load” button.

B.2.6 Channels

Channel	Bias	Flatfield	Binning	Shift	Role	ROI
brite	bias-g	flat-bf	1	0 0	seg	roi
fluor-red	bias-r	flat-flr	1	0 0	qnt	roi
fluor-green	bias-g	flat-flg	1	0 0	qnt	roi
fluor-cyan	bias-b	flat-flb	1	0 0		roi
af-calib	none	none	1	0 0	cal	roiafc

Figure B.3: Channels definition panel in the IMSQT main window.

User specified channel settings. The details for each channel are listed by row. Channel names must correspond to the channels acquired (image subdirectories in the Experiment Path). If a channel line is not used, the channel name must be either blank or “none”.

Correction Channels

Each channel has two correction image channels, “Bias” and “Flatfield”. Bias correction images correct for acquired image noise as well as impart some background correction. Flatfield images correct for non-uniform illumination of the imaging field. The correction channels specified for each imaging channel must exist as image subdirectories in “Correction Path”. Correction frames are generated by averaging the available stack of correction images. If no correction images are available, or channel correction is not needed, the correction channel name may be set to blank or “none”.

Binning

Binning is a pixel neighborhood operation that increases the dynamic range of an image at the cost of reducing image resolution. The value specified in the binning column of

a channel row is the number of pixels that occupy an edge of a square neighborhood. This neighborhood is then averaged to form a new “superpixel” in an output image. For example, a binning value of 2 will average pixels in 2×2 neighborhoods throughout the source image and output an image that is 1/4th the size of the source. This binning operation is in addition to any binning that was carried out during image acquisition. The default binning value is 1, resulting in no change to the source image. Binning is performed before the application of correction frames.

Shift

The shift value the amount (in pixels) to translate an image so that it may be accurately overlaid (registered) with another (typically a segmentation image). This is typically required to compensate for optical shifts incurred when using multiple imaging filters or xy mechanical translation with inherent hysteresis. The value is specified as a vector of x and y pixel values. Image shifting is performed after application of correction frames and software binning.

Role

Specifies the functions that the channel image will perform. Each channel may only have one role.

seg A value of “seg” indicates that a channel will be used for image segmentation. There may be only one channel in the list that has this role.

qnt A value of “qnt” indicates that a channel will be used for segmented object quantification. Multiple channels in the list may have this role (except for the channel defined as

the segmentation channel). If there are multiple channels listed as quantification channels, output data is listed in the order that quantification channels are listed. For instance, if there are two quantification channels, red and yellow, these would correspond to quantified output channels 1 and 2, respectively.

cal A value of “cal” specifies that a channel is to be used for intensity calibration. This feature is not fully implemented and channels labeled with this role are treated as unused channels.

If a particular channel needs to be used for both segmentation and quantification, it may be listed in the channel list twice with different roles for each listing. If a channel is unused its role may be left blank. Unused channel data is not included in output data tables.

ROI

Specifies, by name, the ROI that is to be processed by the segmentation and/or quantification subroutines. Each channel may have only one specified ROI. ROIs named must exist in the ROI list.

B.2.7 Saving and Loading IMSQT sessions

Once experiment details have been entered, it is recommended that it be saved for later retrieval. To do this, go to File → Save. It is best to save the file as “imsqtdata.mat” in the directory specified in “Experiment Path” to avoid confusion with other analysis sessions or other Matlab data files. All data entered into the main window is saved, including data specified in the segmentation, quantification, and tracking panels.

To retrieve a previously saved session, go to File → Open and select the appropriate data file. Upon loading, all fields will be populated with saved values.

B.3 Image segmentation

B.3.1 Automated segmentation

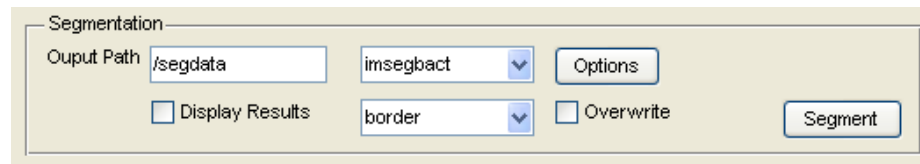


Figure B.4: Segmentation panel in IMSQT main window

Output Path

This specifies the name of the directory in which all segmentation data will be written. This path is relative to the directory specified by “Experiment Path”.

Segmentor

To maintain as much flexibility in the platform, the image segmentation controller was designed to use segmenter modules. The dropdown list to the right of the output path entry field contains a list of available segmentation modules. Modules are stored in,

`<imsqtroot>/bin/segmentors`

and new ones written by the user are automatically added to the dropdown list. Each module is a self contained Matlab function script with the following input and output arguments,

```

function [seg, L, B, W] = segmentor_module(I, seg, ...)
% Segments objects the image I. SEG is a structure that contains
% information regarding the image being segmented with the following
% fields:
5 %
%   imgid      image id (frame number)
%   locid      scan location id
%   channel    channel name
%   roi        region of interest vector (see IMCROP)
10 %   imsize    size of the (cropped) image to be segmented
%
% When the algorithm returns, the following fields will be appended to
% SEG:
%
15 %   objqntidx  indices of object borders to quantify
%   objallidx   indices of all object borders
%
% The algorithm returns:
%
20 %   L        label matrix of segmented objects
%   B        binary object mask of ALL objects (incl. objs on img edge)
%   W        watershed region label matrix
%   seg      segmentation data
%
25 % Optional Arguments are dependent on the specifics of the module and are
% supplied as parameter name / value pairs in a cell array --- e.g.
% segmentor_module(I, seg, 'parameter', value, ...).
% See VARARGIN.
... SEGMENTATION CODE HERE ...

```

In addition to the module script, a graphical options panel, accessible by clicking the “Options” button to the right of the segmentor list must be supplied for each segmentor. Note, this feature may be changed to a text field for increased option specification flexibility in upcoming versions. Currently there are five segmenter modules available,

imsegbf4

Brightfield image segmentation for yeast. Utilizes a hybrid approach that combines the best aspects of imsegbf2 and imsegbf3.

imsegbf3

Brightfield image segmentation for yeast. Utilizes grayscale morphological reconstruction to define objects.

imsegbf2

Brightfield image segmentation for yeast. Utilizes image contrast enhancement to extract object boundaries.

imsegred

Fluorescent image segmentation. Segments small dark objects against a bright background, which is how yeast appear when the local environment is flooded with red fluorescent tracer dye.

imsegbact

Brightfield / phase contrast image segmentation for bacteria. Utilizes simple intensity thresholding for extracting connected pixel regions.

Refer to each module's source code for algorithm details and available input options.

Each segmentor generates a segmentation file for each image in the segmentation channel. These segmentation files are Matlab binary data files (*.mat) stored in the directory specified by the "Output Path". Each file is named according to the stack index it represents.

Segmentation output options

Automated segmentation results may be viewed as they are generated by checking the box next to "Display Results". In addition, the type of display may be chosen as "border" or "filled area" depending on user preferences. Note, displaying segmentation results consumes large amounts of computational overhead and will dramatically slow down the segmentation process. Live display is only recommended for testing purposes on small image sets. Previously generated segmentation results may be overwritten by selecting the checking the box next to "Overwrite". Note, once data has been overwritten, it cannot be

reverted back to its original state.

Running

Automated image segmentation is started by clicking the “Segment” button. A graphical progress meter will appear indicating the estimated amount of time remaining before the segmentation process is complete. Once the segmentation process has been started, it may be canceled by hitting Control-C in the Matlab command window. A more elegant process cancelation method is planned for a future version.

B.3.2 Manual segmentation using SEGEDIT

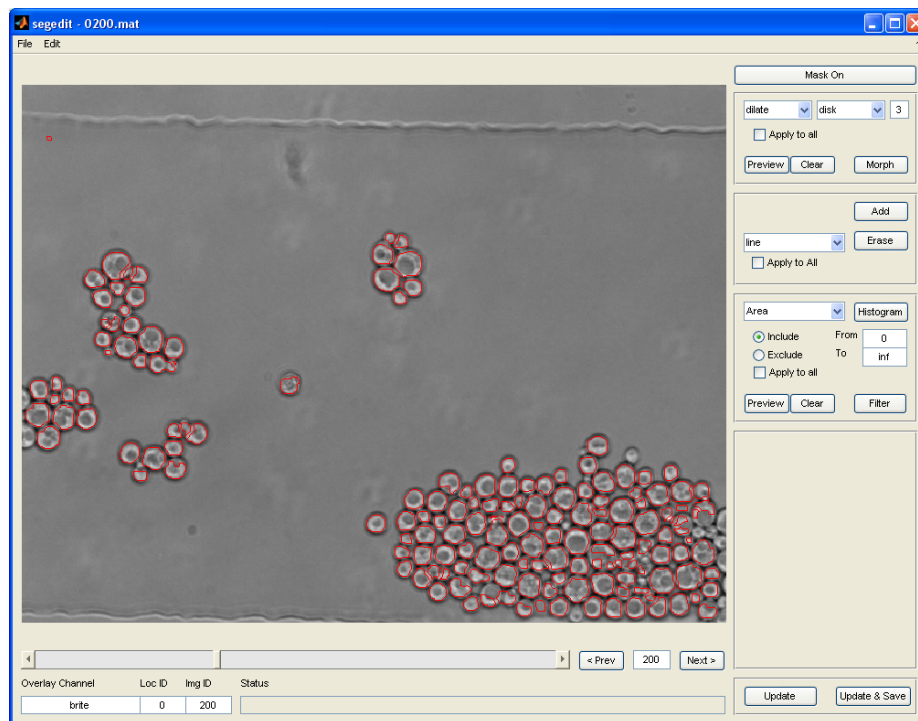


Figure B.5: SEGEDIT manual segmentation editor

No automated segmentation algorithm is 100% accurate. The segmentors provided by IMSQT are at best 95% accurate under ideal imaging conditions. When circumstances require greater accuracy, nothing matches the power of the human eye. SEGEDIT provides the ability to edit the automatically generated segmentation files to suit user specific applications. The editor window consist of a viewing pane, a stack slider, and display, editing, and filtering utilities.

Viewing pane

This is the largest region of the SEGEDIT window. Here it displays the channel image previously designated as the segmentation image along with lines that indicate the borders of segmented objects. Existing objects are displayed using thin red lines. User added objects appear in different colors and line thicknesses. The mask display can be toggled on an off using the mask display button in the upper right hand corner of the window.

Stack slider

This is the scroll bar at the bottom of the SEGEDIT window. Moving the slider cursor allows the user to quickly jump between images in the stack. The user may also specify which image to move to by typing it's index in the text field between the "Prev" and "Next" buttons. Below the stack slider are status indicators which show details of the currently displayed image and whether or not a user requested segmentation process is currently in progress.

Displaying, Editing, and filtering

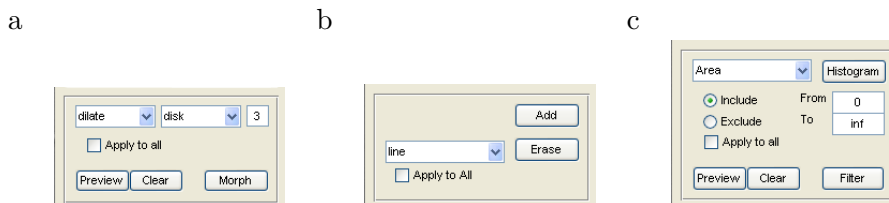


Figure B.6: SEGEDIT subpanels: (a) morphological operations panel, (b) object editing, and (c) morphological filtering.

Along the right hand side of the SEGEDIT window are individual panels that allow the user to control the segmentation display, add/modify segmented objects, and filter objects based on simple morphological properties (Figure B.6). In each panel, if the “Apply to All” checkbox is checked, changes made using each panel will be made to all images in the stack.

Morphological Operations

The basic morphological operations: erode, dilate, open, and close; are available. These are applied to the current image or whole image stack with a disk, square, or diamond structuring element. The dimension of the structuring element, specified in the text field next to the element dropdown list, is in pixels, and modifies the element size in the same way that the `strel` Matlab command generates a structuring element object.

Pressing the “Preview” button will overlay the results of the morphological operation on the currently defined objects in yellow border lines. Pressing “Clear” removes the Preview.

Pressing the “Morph” button applies the operation to the objects in the image. All

applied morphological operations are immediately saved to disk. It is highly recommended that any operation to be performed is previewed to ensure that the desired results will be achieved.

Object Editing

ADDING Object areas may be added to the current image by clicking the “Add” button. This switches the cursor to polygon drawing mode. An object region is added by single clicking at vertices of a polygon and double clicking at the last vertex. The resultant polygon will be displayed in a green outline. To finalize the added object click the “Update” or “Update & Save” buttons. Clicking the “Update” button writes the current object set to memory but not to file. The “Update & Save” button writes the data to file, this step cannot be undone.

SUBTRACTING Object areas may be split or reduced by clicking the “Subtract” button. There are two types of subtractive modes, the default “Line” subtraction, and “Area” (polygon) subtraction. “Line” subtraction is most commonly used to split touching objects. Subtraction lines are drawn in the same way as polygons are (by selecting vertex points) except that the resultant object is not closed. In addition, subtraction lines are drawn as thick blue lines.

“Area” subtraction is similar to adding objects, except that any objects that are in contact (overlapping) as well as objects enclosed by the subtractive polygon are removed on update. This is useful if “Apply All” is selected for removing persistent unwanted objects from the image. Subtractive polygons are drawn in thin blue lines. Global subtractive areas are drawn in thin light blue lines.

DELETING Any object may be selected when not in polygon or line drawing mode.

Once highlighted, objects may be removed by simply hitting the DELETE key. The change is not permanent until the image is updated and saved.

Filtering

The objects may be filtered based on several morphological parameters and property values: area, perimeter, intensity, and variance. Clicking on the “histogram” button will display a distribution of the parameter selected in the drop down box over the objects in the currently displayed image. To filter objects, specify the range of values in the text fields next to “From” and “To”. The values entered are inclusive. Select “Include” to keep, or “Exclude” to remove, objects that fall within the parameter range. As with morphological operations, the filtering process can be “Previewed”. Objects that will remain will be highlighted with yellow border lines. The “Clear” button turns off the preview display. The “Filter” button starts the filtration process. Use the “Apply All” check box to specify if the filtering operation is to be applied to the entire image set.

Shortcut Keys

Key(s)	Function
Delete, or Numpad- Decimal	Deletes the currently selected object
→, Space, or Numpad-6	Advances the stack one image
←, Backspace, or Numpad-4	Reverses the stack one image
+	Enters object addition mode
-	Enters object subtraction mode using current subtractive object type
F12 or Numpad-Enter	Updates the current image mask and saves it to file
m or Numpad- *	Toggles mask display

B.4 Object Quantification

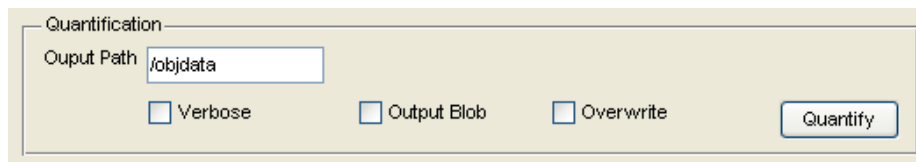


Figure B.7: Object quantification panel in IMSQT main window

Output Path

This specifies where the object data will be written. The path specified is relative to the path specified in “Experiment Path”. Object data is written in plain ASCII formatted files and named according to image index. Each object data file contains the position and channel quantification data for each segmented object in the image. Each object is assigned a unique identification number on a per image basis and has the following data associated with it,

Column	Description
CENTX	x position of the object centroid
CENTY	y position of the object centroid
BBOXX	x position of the lower left corner of the object bounding box
BBOXY	y position of the lower left corner of the object bounding box
BBOXW	width of the object bounding box
BBOXH	height of the object bounding box
AREA	euclidian area of the object
PERIMNATIVE	euclidian perimeter of the object (includes object holes)
PERIMFILLED	perimeter of the filled object (excludes object holes)
PERIMCONVEX	perimeter of the object's convex hull
MEAN1	channel 1 mean intensity within the object
STD1	channel 1 pixel intensity standard deviation within the object
MIN1	channel 1 minimum intensity value within the object
MAX1	channel 1 maximum intensity value within the object
BGLVL1	channel 1 background level (measured in non-object regions)
...	...
MEANn	channel n mean intensity within the object
STDn	channel n pixel intensity standard deviation within the object
MINn	channel n minimum intensity value within the object
MAXn	channel n maximum intensity value within the object
BGLVLn	channel n background level (measured in non-object regions)

Output Options

Verbose

If this option is selected, data that will be saved to image object data files will also be output to the Matlab command window. It is recommended that this be used only for debugging purposes or on small image sets since this significantly increases processing overhead, increasing the amount of time the procedure will take.

Output Blob

If this option is selected, channel quantification parameters will be averaged over all objects on a per image basis and output to a file named `<locid>.blob.dat`. Note, object position and bounding box data is omitted from this data file.

Overwrite

If this option is selected, previously existing object data will be overwritten. This cannot be undone.

Running

Image object quantification is started by clicking the “Quantify” button. A graphical progress meter will appear indicating the estimated amount of time remaining before the quantification process is complete. Once the quantification process has been started, it may be canceled by hitting Control-C in the Matlab command window. A more elegant process cancelation method is planned for a future version. Note, blob output will not be generated if the quantification process is not completed.

B.5 Object Tracking

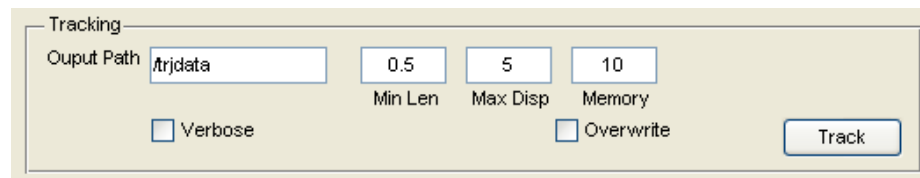


Figure B.8: Object tracking panel in IMSQT main window

The tracking algorithm used by this application is a Matlab port of the algorithm developed by Crocker and Grier[24]. I have made minor modifications to the source code improving parameter specification and using native Matlab functions where possible. The original port may be found at:

<http://www.deas.harvard.edu/projects/weitzlab/matlab/>

The tracking algorithm is capable of locating and descrambling objects based on multiple parameters. For this application, only the centroid location is utilized. Future versions will provide the option to include channel intensity and object area as descrambling parameters.

Output Path

This specifies where the individual object data will be written. The path specified is relative to the path specified in “Experiment Path”. Object data is written in plain ASCII formatted files and is organized into two directories: **byobj** and **byimg**. Files stored in the **byobj** directory are named according to unique object identifiers and contain data over all images for only the object id specified. Files stored in the **byimg** directory are named according to image index and contain data for all objects in each image.

Note, the tracking algorithm reassigns object identifiers and persists them for the lifetime of the object. In addition, data columns are truncated to object position (centroid location), area, and image intensity properties (mean intensity, standard deviation, and background level).

Input Options

Min Len

Specifies the minimum length of valid output trajectories. If this value is > 0 and < 1 it is interpreted as a percentage of the longest possible trajectory (the length of the image set). If this value is ≥ 1 it is interpreted as an absolute point length — e.g. the trajectory must have N specified points in its trajectory to be valid. Invalid trajectories are discarded.

Max Disp

Specifies the maximum displacement in pixels that an object may move between images. Note, this parameter is very sensitive. If this is set too high, the algorithm will encounter “difficult combinatorics” and exit with an error.

Memory

Specifies the number of images that an object may be “lost” and subsequently reacquired. Note this may result in object mislabeling if there are frequent losses and object relocations, increasing the work needed for trajectory validation.

Output Options

Verbose

Displays tracking algorithm progress in the Matlab command window. It is recommended that this option be selected since the program has not been fully integrated with IMSQT’s graphical progress indicators.

Overwrite

Overwrites existing tracking data.

Running

Image object tracking is started by clicking the “Track” button. If verbose output is selected, tracking progress will appear in the Matlab command window. Otherwise, the system will maintain a “Busy” status until the algorithm finishes or exits with an error. Once the tracking process has been started, it may be canceled by hitting Control-C in the Matlab command window. Note, tracking output will not be generated if the process is canceled or exits with an error. In addition, changing input options may change output object identifiers. If data exists from a previous tracking run, it is recommended that it be deleted manually before tracking objects again.

B.5.1 Tracking Validation with TRACKEDIT

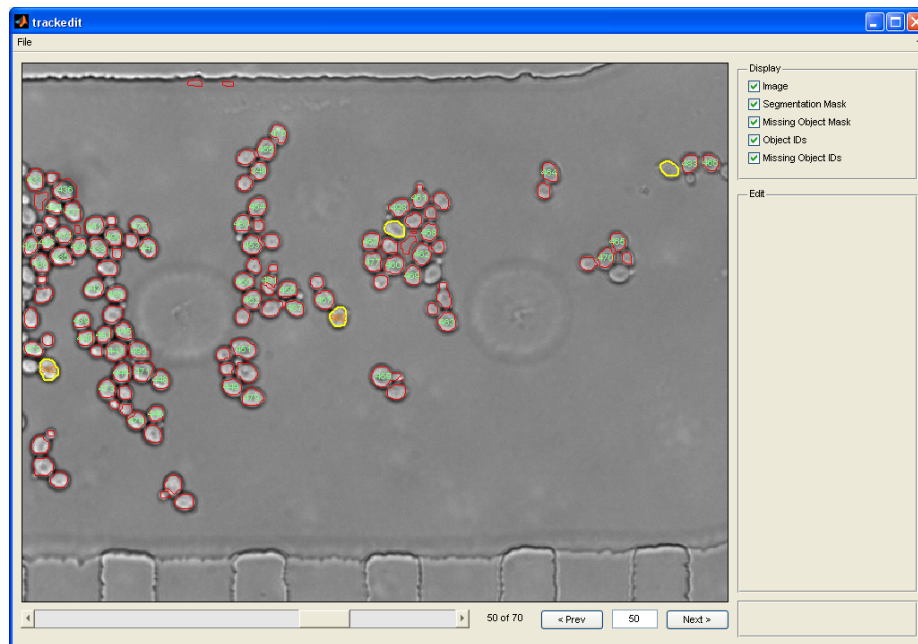


Figure B.9: TRACKEDIT object tracking validation viewer

The TRACKEDIT (pronounced TRACK-EDIT) utility allows users to validate automatically generated trajectories. Like the SEGEDIT utility, it has a main viewing window which displays an image with a segmentation overlay and a stack slider that allows the user to move freely through the image set.

Starting a TRACKEDIT session

TRACKEDIT is started by typing `trackedit` at the Matlab command prompt. TRACKEDIT requires two sets of data,

- segmentation data generated by automated and/or manual means
- tracked object data on a per image basis, generated by the automated tracking algorithm

This information is stored within the session data file create by the main IMSQT window (e.g. `imsqtdata.mat`) and should be loaded accordingly for a tracking dataset that requires validation.

Object display

While similar to SEGEDIT, the display method is geared more toward identifying mislabeled and “lost” objects. As in SEGEDIT, the primary segmentation masks are shown as red outlines around cells.

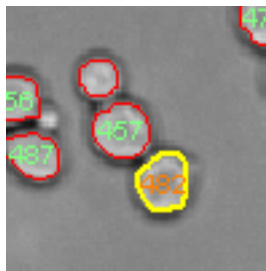


Figure B.10: A sample of TRACKEDIT object display.

If an object satisfies the trajectory length criteria, its unique identification number will be displayed in green at its centroid location. If an object has disappeared relative to the previous frame, a yellow outline where the object was expected to be will be shown. If the missing object satisfies the trajectory length criteria and is still in “memory”, its unique identifier will be shown in orange. Every aspect of the display can be toggled on or off using the display control panel.

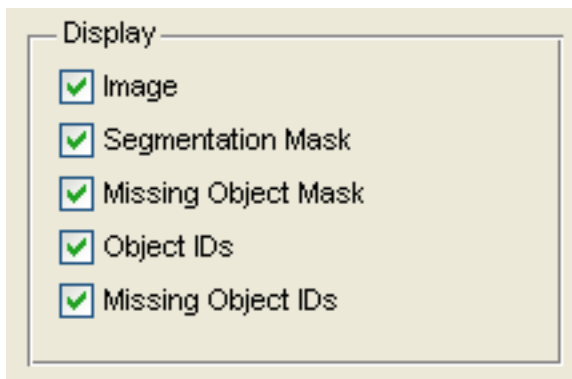


Figure B.11: The TRACKEDIT display control panel.

Shortcut Keys

Key(s)	Function
→, Space, or Numpad-6	Advances the stack one image
←, Backspace, or Numpad-4	Reverses the stack one image

Validating trajectories

To validate a trajectory, a user must visually inspect that the object maintains its unique identification number throughout its lifetime in the image stack (from object appearance to disappearance). Often objects will change ID numbers part way through a stack either due to a gross shift in object positions, “lost” frames greater than the number of frames allowed by the “memory” setting, or both. Under these circumstances it is necessary to generate an object trajectory link table that associates new object IDs with the originating object ID. This table is generated by hand, and is used by TRACKVIEW via the subalgorithm TRACKLINK to produce extended single-cell trajectories.

The format of the trajectory link table is shown below.

IMGID	OBJID
0	<objid>
:	:
<ending_imgid>	<ending_objid>

Note, the column headers must be included in the table. The table is formatted in plain ASCII and is tab delimited. The linking behavior depends on the value of OBJID,

<int> The object was reassigned a new object id <int> at the specified image id.

-1 The tracking algorithm lost the object at the specified image id. The next row in the link table specifies when the object returns and the object’s new object id (if reassigned). Data between the specified image id values is skipped. This functionality is

Table B.1: A sample trajectory link table.

IMGID	OBJID
48	1308
263	-1
265	2461
288	#

somewhat redundant with specifying a “memory” value when running the automated tracking algorithm. However this provides more flexibility by allowing for variable length “lost” frame intervals, and more accurate object recovery upon reappearance.

The object was completely lost at the specified image id. This effectively terminates the objects trajectory. If the object persists to the last image in the stack the last row of the link table should list the last image id with a “#” as the object id value.

A sample link table is shown below.

Here, the object first appeared at frame 48 as object 1308. It maintained this ID until frame 263 when it was lost until frame 265 where it was recovered as object 2461. The object was completely lost at frame 288.

Link tables should be stored in `/trjdata/<locid>/byobj/lnk`. This is the default location recognized by the TRACKVIEW utility. Each file should be named `<original obj id>_<ending obj id>.txt` with object IDs padded to four places with zeros. Thus the link table file for the data above would be named `0048_2461.txt`. The first number in the name is what associates the link table to the correct object when called from within TRACKVIEW. While not currently directly used by any trajectory utilities, it is recommended that the terminating id and file extension of a trajectory link table file be formatted accordingly to maintain readability.

B.5.2 Trajectory Viewing with TRACKVIEW

Once trajectories are validated and appropriately linked, they can be viewed using the TRACKVIEW command line utility. From within Matlab, navigate to the directory where the `imsqtdata.mat` file for the imaging session is located.

Options

link

A boolean value that determines if the algorithm should apply a link table to the object trajectory requested. The default value is false. Note, this option may be replaced with an auto-detection method in future versions.

trjpath

A string value that specifies the location of the object tracking data relative to the current directory. The default value is `"/trjdata"`.

plots

A cell array specifying the plots to display. The default value is `{'f12', 'area', 'var2'}`. Available plots are:

`f1n` fluorescence (mean intensity) for channel n versus time.

`stdn` pixel standard deviation for channel n versus time.

`bgn` background level for channel n versus time.

`varn` pixel variance for channel n versus time.

`area` object area versus time.

`perim` object perimeter versus time.

xy object centroid pixel position (x versus y).

Note, in graphic mode (see below) only the first 4 plots requested will be displayed, however, an unlimited number of plot data sets may be requested.

smoothing

A cell array specifying if and how data should be smoothed using the RLOWESS smoothing method. Smoothing definitions are:

lg large window smoothing.

sm small window smoothing.

dt detrended output (small window smoothing - large window smoothing)

” (matlab empty string) no smoothing for displayed data.

Smoothing requires that the `smival` and `imival` be defined. The base window size is the value specified by `smival` divided by the value specified by `imival`. Large smoothing windows are $2\times$ the base window size. Small smoothing windows are $1/2$ the base window size.

smival

The nominal smoothing interval in data time units — e.g. if the data is recorded in minutes and an hour long smoothing interval is desired, this value should be set to 60.

imival

The nominal time interval between data points in data time units. The default interval is 5 minutes.

color

A Matlab color specification vector defining what color displayed lineseries will be.

Default is blue ([0 0 1]).

graphic

A boolean value specifying if plots should be generated. The default value is true.

Note, TRACKVIEW also outputs the data requested in a cell array if more specialized plotting is required.

Examples

Here is the default plots for the trajectory shown in Table B.1.

```
>> trackview(0, 1308);
```

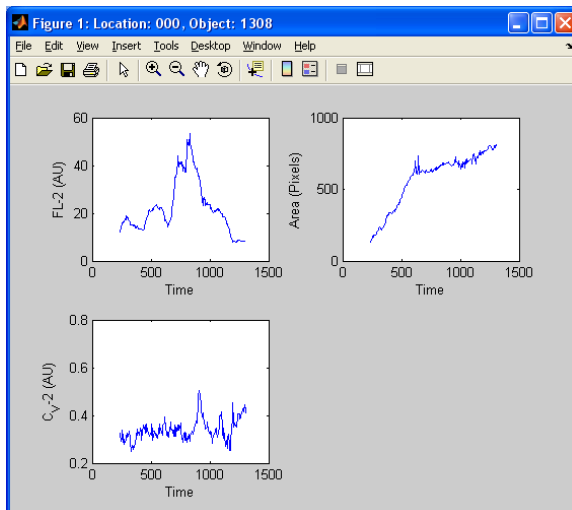


Figure B.12: Default graphical output using TRACKVIEW.

Here are the plotting results using defined plot output and smoothing.

```
>> trackview(0, 1308, 'plots', {'f12','f12','f12','f12'}, ...
'smoothing', {'', 'sm', 'lg', 'dt'}, 'smival', 60);
```

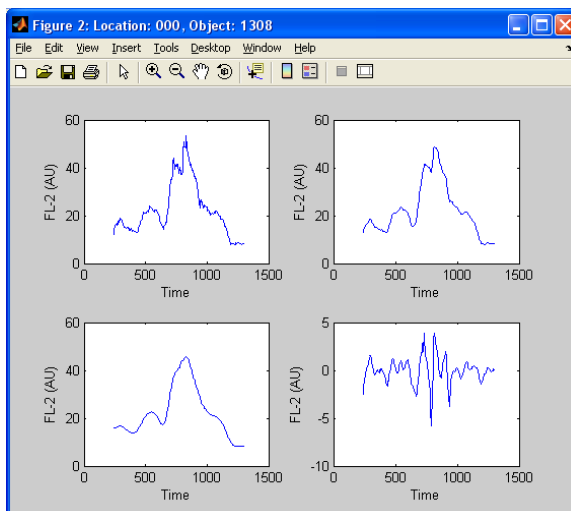


Figure B.13: Data smoothing using TRACKVIEW.

An example of non-graphical output.

```
>> [f data] = trackview(0, 1308, 'graphic', false);
>> data

data =
5
    [211x1 double]    [211x1 double]    'Time'    'FL-2 (AU)'
    [211x1 double]    [211x1 double]    'Time'    'Area (Pixels)'
    [211x1 double]    [211x1 double]    'Time'    'C_V-2 (AU)'

10 >>
```

The columns of the output cell array are “x-values”, “y-values”, “x-label”, “y-label”, respectively.

B.6 Complete source code for IMSQT main window

```

function varargout = imsqt(varargin)
% IMSQT M-file for imsqt.fig
%     IMSQT, by itself, creates a new IMSQT or raises the existing
%     singleton*.
5 %
%     H = IMSQT returns the handle to a new IMSQT or the handle to the
%     existing singleton*.
%
%     IMSQT('CALLBACK',hObject,eventData,handles,...) calls the local
10 %     function named CALLBACK in IMSQT.M with the given input arguments.
%
%     IMSQT('Property','Value',...) creates a new IMSQT or raises the
%     existing singleton*. Starting from the left, property value pairs
%     are applied to the GUI before imsqt_OpeningFunction gets called. An
15 %     unrecognized property name or invalid value makes property
%     application stop. All inputs are passed to imsqt_OpeningFcn via
%     varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
20 %     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help imsqt
25
% Last Modified by GUIDE v2.5 14-Apr-2006 13:15:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
30 gui_State = struct('gui_Name',       mfilename, ...
                    'gui_Singleton',  gui_Singleton, ...
                    'gui_OpeningFcn', @imsqt_OpeningFcn, ...
                    'gui_OutputFcn',  @imsqt_OutputFcn, ...
                    'gui_LayoutFcn',  [] , ...
35                    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

40 if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
45 % End initialization code - DO NOT EDIT

% --- Executes just before imsqt is made visible.
function imsqt_OpeningFcn(hObject, eventdata, handles, varargin)
50 % This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to imsqt (see VARARGIN)

```

```

55 % Choose default command line output for imsqt
handles.output = hObject;

% set application defaults
60 handles.info = struct( ...
    'path',          '', ...
    'corrpath',     '', ...
    'prefix',       '', ...
    'id',           '', ...
65    'locid',        0, ...
    'indices',      [0:1:0], ...
    'indicesstr',   '0:1:0' ...
    );

70 handles.channelgrid = handles.activex3;
% initialize the channels editor;
% default channels:
roi = []; roiafc = [];
75 csColName      = {
    'Channel', 'Bias', 'Flatfield', 'Binning', 'Shift', 'Role', 'ROI';
    };
csColType        = {
    'string', 'string', 'string', 'numeric', 'numeric', 'string', 'string';
80    };
handles.channelgridcols = csColName;
handles.channelgridcoltypes = csColType;
cvChannels       = {
    'brite',      'bias-g', 'flat-bf', 1, [0, 0], 'seg', 'roi';
85    'fluor-red', 'bias-r', 'flat-flr', 1, [0, 0], 'qnt', 'roi';
    'fluor-green', 'bias-g', 'flat-flg', 1, [0, 0], 'qnt', 'roi';
    'fluor-cyan', 'bias-b', 'flat-flb', 1, [0, 0], '', 'roi';
    'af-calib',   'none',   'none',    1, [0, 0], 'cal', 'roiafc';
    };
90 imsqt_fillgrid(handles.channelgrid, csColName, cvChannels)
handles.channels = imsqt_parsechannelgrid(handles);

handles.roigrid = handles.activex4;
% initialize the roi editor
95 csColName      = {
    'Name', 'xmin', 'ymin', 'width', 'height';
    };
csColType        = {
    'string', 'numeric', 'numeric', 'numeric', 'numeric';
100    };
handles.roigridcols = csColName;
handles.roigridcoltypes = csColType;
cvROIs           = {
    'roi'      , 'auto', 'auto', 'auto', 'auto';
105    'roiafc', 'auto', 'auto', 'auto', 'auto';
    };
imsqt_fillgrid(handles.roigrid, csColName, cvROIs)
handles.rois = imsqt_parseroigrid(handles.roigrid);

110 % segmentation panel defaults
handles.segment.path = '/segdata';

```

```

handles.segment.algorithm = 'imsegbf2';
% initialize segmenter options to nothing ... this will be populated with
% defaults if the user does not specify anything using the segopts gui.
115 handles.segment.opts = {};
    handles.segment.optsdef = {...
        'thresh',          0.4, ...
        'smoothkernels',  [2 15], ...
        'objradius',      [2 15], ...
120     'area',             [0 inf], ...
        'eccentricity',    [0 inf], ...
        'solidity',        [0 1], ...
        'intensity',       [], ...
    };
125
    handles.segment.display = false;
    handles.segment.displaymethod = 'border';
    handles.segment.override = false;

130 % quantification panel defaults
    handles.quant.path = '/objdata';
    handles.quant.verbose = false;
    handles.quant.blobout = false;
    handles.quant.override = false;
135
    % tracking panel defaults
    handles.track.path = '/trjdata';
    handles.track.minlength = 0.5;
    handles.track.maxdisp = 5;
140 handles.track.mem = 10;
    handles.track.verbose = false;
    handles.track.override = false;

    % data for doing file loading and saving
145 handles.uidata.appname = get(handles.figure1, 'name');
    handles.uidata.file = '';
    handles.uidata.cwd = cd;

    % populate the segmentors list
150 p = mfilename('fullpath');
    p = fileparts(p);
    stSegmentors = dir([p filesep 'bin' filesep 'segmentors' filesep '*.m']);
    sSegmentors = cell(length(stSegmentors), 1);
    for i = 1:length(stSegmentors),
155     [fpath, fname, fext] = fileparts(stSegmentors(i).name);
        sSegmentors{i} = fname;
    end;
    set(handles.pmnSegSegmenter, 'String', sSegmentors);

160 % Update handles structure
    guidata(hObject, handles);

    % UIWAIT makes imsqt wait for user response (see UIRESUME)
    % uiwait(handles.figure1);
165

    % --- Outputs from this function are returned to the command line.
    function varargout = imsqt_OutputFcn(hObject, eventdata, handles)

```

```

% varargout cell array for returning output args (see VARARGOUT);
170 % hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
175 varargout{1} = handles.output;

function edtExptPath_Callback(hObject, eventdata, handles)
180 % hObject handle to edtExptPath (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtExptPath as text
185 % str2double(get(hObject,'String')) returns contents of edtExptPath
% as a double
handles.info.path = get(hObject, 'string');
guidata(hObject, handles);

190 % --- Executes during object creation, after setting all properties.
function edtExptPath_CreateFcn(hObject, eventdata, handles)
% hObject handle to edtExptPath (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called
195
% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
    get(0,'defaultUicontrolBackgroundColor'))
200 set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pbExptPathBrowse.
205 function pbExptPathBrowse_Callback(hObject, eventdata, handles)
% hObject handle to pbExptPathBrowse (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
sDir = uigetdir;
210 if sDir ~= 0,
    set(handles.edtExptPath, 'string', sDir);
end;

215 function edtSegDataPath_Callback(hObject, eventdata, handles)
% hObject handle to edtSegDataPath (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

220 % Hints: get(hObject,'String') returns contents of edtSegDataPath as text
% str2double(get(hObject,'String')) returns contents of
% edtSegDataPath as a double
handles.segment.path = get(hObject, 'string');
guidata(hObject, handles);
225

```

```

% --- Executes during object creation, after setting all properties.
function edtSegDataPath_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtSegDataPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
230 % handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
235         get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

240
function edtObjDataPath_Callback(hObject, eventdata, handles)
% hObject    handle to edtObjDataPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
245
% Hints: get(hObject,'String') returns contents of edtObjDataPath as text
%       str2double(get(hObject,'String')) returns contents of
%       edtObjDataPath as a double
handles.quant.path = get(hObject, 'String');
250 guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edtObjDataPath_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtObjDataPath (see GCBO)
255 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
260 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

265

function edtTrjDataPath_Callback(hObject, eventdata, handles)
% hObject    handle to edtTrjDataPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
270 % handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtTrjDataPath as text
%       str2double(get(hObject,'String')) returns contents of
%       edtTrjDataPath as a double
275

% --- Executes during object creation, after setting all properties.
function edtTrjDataPath_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtTrjDataPath (see GCBO)
280 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
285 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

290

function edtExptID_Callback(hObject, eventdata, handles)
% hObject    handle to edtExptID (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
295 % handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtExptID as text
% str2double(get(hObject,'String')) returns contents of edtExptID as
% a double
300

% --- Executes during object creation, after setting all properties.
function edtExptID_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtExptID (see GCBO)
305 % eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
310 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

315

function edtExptPrefix_Callback(hObject, eventdata, handles)
% hObject    handle to edtExptPrefix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
320 % handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtExptPrefix as text
% str2double(get(hObject,'String')) returns contents of
% edtExptPrefix as a double
325

% --- Executes during object creation, after setting all properties.
function edtExptPrefix_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtExptPrefix (see GCBO)
330 % eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
335 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
end

```



```

340
function edtLocID_Callback(hObject, eventdata, handles)
% hObject    handle to edtLocID (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
345 % handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtLocID as text
%        str2double(get(hObject,'String')) returns contents of edtLocID as
%        a double
350

% --- Executes during object creation, after setting all properties.
function edtLocID_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtLocID (see GCBO)
355 % eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
360 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

365

function edtIdxStart_Callback(hObject, eventdata, handles)
% hObject    handle to edtIdxStart (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
370 % handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtIdxStart as text
%        str2double(get(hObject,'String')) returns contents of edtIdxStart
%        as a double
375

% --- Executes during object creation, after setting all properties.
function edtIdxStart_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtIdxStart (see GCBO)
380 % eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
385 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

390

function edtIdxStep_Callback(hObject, eventdata, handles)
% hObject    handle to edtIdxStep (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
395 % handles   structure with handles and user data (see GUIDATA)

```

```

% Hints: get(hObject,'String') returns contents of edtIdxStep as text
%         str2double(get(hObject,'String')) returns contents of edtIdxStep
%         as a double
400

% --- Executes during object creation, after setting all properties.
function edtIdxStep_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtIdxStep (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
405 % handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
410 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

415

function edtIdxStop_Callback(hObject, eventdata, handles)
% hObject    handle to edtIdxStop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
420 % handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtIdxStop as text
%         str2double(get(hObject,'String')) returns contents of edtIdxStop
%         as a double
425

% --- Executes during object creation, after setting all properties.
function edtIdxStop_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtIdxStop (see GCBO)
430 % eventdata  reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
435 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

440

% --- Executes on button press in pbSegment.
function pbSegment_Callback(hObject, eventdata, handles)
% hObject    handle to pbSegment (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
445 % handles   structure with handles and user data (see GUIDATA)

if ~isempty(imsqt_getexptpath(handles)),
    handles = imsqt_collectinfo(handles);
    if isempty(handles.segment.opts) || ~iscell(handles.segment.opts),
450         handles.segment.opts = handles.segment.optsdef;
    end;

    handles.channels = imsqt_parsechannelgrid(handles);

```

```

handles.rois = imsqt_parseroigrid(handles.roigrid);
455
bProceed = true;
%   set(hObject, 'enable', 'off'); drawnow;
if handles.segment.overwrite,
    btn = questdlg(...
460        ['The settings you have chosen will replace any existing data.'...
        ' Do you wish to proceed?'], ...
        'Overwrite Data?', 'Yes', 'No', 'No');
    bProceed = strcmpi(btn, 'Yes');
end;
465 if bProceed,
    imsegauto(handles.info, ...
              handles.segment, ...
              handles.channels, ...
              handles.rois)
470 end;
%   set(hObject, 'enable', 'on'); drawnow;

guidata(hObject, handles);
else
475 errordlg('Experiment path information not specified!', 'Error');
end;

% --- Executes on button press in pbQuantify.
function pbQuantify_Callback(hObject, eventdata, handles)
480 % hObject    handle to pbQuantify (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
if ~isempty(imsqt_getexptpath(handles)),
    handles = imsqt_collectinfo(handles);
485 if isempty(handles.segment.opts),
    handles.segment.opts = handles.segment.optsdef;
end;

handles.channels = imsqt_parsechannelgrid(handles);
490 handles.rois = imsqt_parseroigrid(handles.roigrid);

bProceed = true;
%   set(hObject, 'enable', 'off'); drawnow;
if handles.quant.overwrite,
495    btn = questdlg(...
        ['The settings you have chosen will replace any existing data.'...
        ' Do you wish to proceed?'], ...
        'Overwrite Data?', 'Yes', 'No', 'No');
    bProceed = strcmpi(btn, 'Yes');
500 end;
if bProceed,
    imsegcollect(handles.info, ...
                 handles.segment, ...
                 handles.quant, ...
505                 handles.channels, ...
                 handles.rois);
end;
%   set(hObject, 'enable', 'on'); drawnow;

510 guidata(hObject, handles);

```

```

else
    errordlg('Experiment path information not specified!', 'Error');
end;

515 function edtTrjDatapath_Callback(hObject, eventdata, handles)
    % hObject    handle to edtTrjDatapath (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
520
    % Hints: get(hObject,'String') returns contents of edtTrjDatapath as text
    %         str2double(get(hObject,'String')) returns contents of
    %         edtTrjDatapath as a double
    handles.track.path = get(hObject, 'string');
525 guidata(hObject, handles);

    % --- Executes during object creation, after setting all properties.
    function edtTrjDatapath_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to edtTrjDatapath (see GCBO)
530 % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: edit controls usually have a white background on Windows.
    %         See ISPC and COMPUTER.
535 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

540
    % --- Executes on button press in pbTrack.
    function pbTrack_Callback(hObject, eventdata, handles)
    % hObject    handle to pbTrack (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
545 % handles    structure with handles and user data (see GUIDATA)
    if ~isempty(imsqt_getexptpath(handles)),
        handles = imsqt_collectinfo(handles);

        handles.channels = imsqt_parsechannelgrid(handles);

550
        bProceed = true;
        % set(hObject, 'enable', 'off'); drawnow;
        if handles.track.overwrite,
            btn = questdlg(...
555             ['The settings you have chosen will replace any existing data.'...
                ' Do you wish to proceed?'], ...
                'Overwrite Data?', 'Yes', 'No', 'No');
            bProceed = strcmpi(btn, 'Yes');
        end;
560     if bProceed,
            imsegtrack(handles.info, ...
                handles.quant, ...
                handles.track, ...
                handles.channels);
565     end;
    % set(hObject, 'enable', 'on'); drawnow;

```

```

        guidata(hObject, handles);
    else
570     errordlg('Experiment path information not specified!', 'Error');
    end;

    % --- Executes on selection change in pmnSegSegmenter.
    function pmnSegSegmenter_Callback(hObject, eventdata, handles)
575 % hObject     handle to pmnSegSegmenter (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles     structure with handles and user data (see GUIDATA)

    % Hints: contents = get(hObject,'String') returns pmnSegSegmenter contents
580 %           as cell array contents{get(hObject,'Value')} returns selected item
    %           from pmnSegSegmenter

    contents = get(hObject, 'string');
    handles.segment.algorithm = contents{get(hObject, 'value')};
585 guidata(hObject, handles);

    % --- Executes during object creation, after setting all properties.
    function pmnSegSegmenter_CreateFcn(hObject, eventdata, handles)
    % hObject     handle to pmnSegSegmenter (see GCBO)
590 % eventdata reserved - to be defined in a future version of MATLAB
    % handles     empty - handles not created until after all CreateFcns called

    % Hint: popupmenu controls usually have a white background on Windows.
    %           See ISPC and COMPUTER.
595 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

600
    % --- Executes on button press in pbSegOpts.
    function pbSegOpts_Callback(hObject, eventdata, handles)
    % hObject     handle to pbSegOpts (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
605 % handles     structure with handles and user data (see GUIDATA)

    handles.segment.opts = imsqt_segopts('segmenter', ...
        handles.segment.algorithm);
    guidata(hObject, handles);

610
    % --- Executes on button press in chkQntVerbose.
    function chkQntVerbose_Callback(hObject, eventdata, handles)
    % hObject     handle to chkQntVerbose (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
615 % handles     structure with handles and user data (see GUIDATA)

    % Hint: get(hObject,'Value') returns toggle state of chkQntVerbose
    handles.quant.verbose = get(hObject, 'value');
    guidata(hObject, handles);

620
    % --- Executes on button press in chkDispSeg.
    function chkDispSeg_Callback(hObject, eventdata, handles)
    % hObject     handle to chkDispSeg (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB

```

```

625 % handles      structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of chkDispSeg

630 % --- Executes on selection change in pmnDispMethod.
function pmnDispMethod_Callback(hObject, eventdata, handles)
% hObject      handle to pmnDispMethod (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
635
% Hints: contents = get(hObject,'String') returns pmnDispMethod contents as
%         cell array contents{get(hObject,'Value')} returns selected item
%         from pmnDispMethod

640
% --- Executes during object creation, after setting all properties.
function pmnDispMethod_CreateFcn(hObject, eventdata, handles)
% hObject      handle to pmnDispMethod (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
645 % handles      empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
650         get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

655
function edtMinTrajLen_Callback(hObject, eventdata, handles)
% hObject      handle to edtMinTrajLen (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
660
% Hints: get(hObject,'String') returns contents of edtMinTrajLen as text
%         str2double(get(hObject,'String')) returns contents of
%         edtMinTrajLen as a double

665
% --- Executes during object creation, after setting all properties.
function edtMinTrajLen_CreateFcn(hObject, eventdata, handles)
% hObject      handle to edtMinTrajLen (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
670 % handles      empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
675         get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

680

```

```

% --- Executes on button press in pbSelectROI.
function pbSelectROI_Callback(hObject, eventdata, handles)
% hObject    handle to pbSelectROI (see GCBO)
685 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% select the roi from the imaging channel specified.  user supplies name of
% roi after selection is complete.
690
sExptPath = imsqt_getexptpath(handles);
if ~isempty(sExptPath),
    sChName = get(handles.edtROIDefCh, 'string');
    iChImgID = str2num(get(handles.edtROIDefIdx, 'string'));
695    iLocID = str2num(get(handles.edtLocID, 'string'));
    I = getimg(sChName, iLocID, iChImgID, 'basepath', sExptPath);

    hFigCrop = figure;
    [I, roidata] = imcrop(imadjust(I));
700    close(hFigCrop);
    clear I hFigCrop;

    roiname = inputdlg('ROI name:', 'Specify ROI', 1, {'roi'});

705    handles.rois = imsqt_roiaddreplace(handles.rois, ...
                                       {roiname{1}, roidata});
    imsqt_roigridupdate(handles);
    guidata(hObject, handles);
else,
710    errordlg('All experiment path fields must be specified!', 'Error');
end;

% -----
715 function sExptPath = imsqt_getexptpath(handles)
% sExptPath = [];
% csStrings = get([handles.edtExptPath, ...
%                 handles.edtExptPrefix, ...
%                 handles.edtExptID], 'string');
720 %
% % only assign the path if all the fields have been set
% if isempty(strmatch('', csStrings, 'exact')),
%     sExptPath = sprintf('%s%s%s', csStrings{1}, filesep, csStrings{2:3});
% end;
725 sExptPath = get(handles.edtExptPath, 'string');
if isempty(sExptPath),
    sExptPath = [];
end;

730

function edtROIDefCh_Callback(hObject, eventdata, handles)
% hObject    handle to edtROIDefCh (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
735 % handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtROIDefCh as text
%        str2double(get(hObject,'String')) returns contents of edtROIDefCh

```

```

%           as a double
740

% --- Executes during object creation, after setting all properties.
function edtROIDefCh_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtROIDefCh (see GCBO)
745 % eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%           See ISPC and COMPUTER.
750 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

755

function edtROIDefIdx_Callback(hObject, eventdata, handles)
% hObject    handle to edtROIDefIdx (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
760 % handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtROIDefIdx as text
%         str2double(get(hObject,'String')) returns contents of edtROIDefIdx
%         as a double
765

% --- Executes during object creation, after setting all properties.
function edtROIDefIdx_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtROIDefIdx (see GCBO)
770 % eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%           See ISPC and COMPUTER.
775 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

780

% -----
785 function imsqt_fillgrid(hGrid, csCols, cvValues)
    for c = 1:size(cvValues, 2),
        for r = 0:size(cvValues, 1),
            set(hGrid, 'row', r, 'col', c-1);
            if r == 0,
790                 % set column headers
                    set(hGrid, 'text', csCols{c});
            else
                val = cvValues{r, c};
                if isstr(val)
795                     set(hGrid, 'text', val);
                end
            end
        end
    end

```



```

        else
            set(hGrid, 'text', num2str(val));
        end;
    end;
800 end;
    end;
    set(hGrid, 'row', 1, 'col', 0);

805 % -----
    function cvChannels = imsq_t_parsechannelgrid(handles)
        hGrid = handles.channelgrid;
        csColType = handles.channelgridcoltypes;

810 R = get(hGrid, 'Rows');
        C = get(hGrid, 'Cols');
        cvChannels = cell(R-1, C);

        for r = 1:R-1,
815     for c = 0:C-1,
            set(hGrid, 'row', r, 'col', c);
            sTxt = get(hGrid, 'text');

            %           printf('%12s%3d%3d', sTxt, bIsStr, bIsNum)
820     switch csColType{c+1},
                case 'string'
                    vVal = sTxt;
                case 'numeric'
                    vVal = str2num(sTxt);
825     otherwise,
                    % hopefully it's impossible to get here.
                    vVal = [];
                end;
            cvChannels{r, c+1} = vVal;
830     end;
        end;
        set(hGrid, 'row', 1, 'col', 0);

835 % -----
    function cvROIs = imsq_t_parseroigrid(hGrid)
        R = get(hGrid, 'Rows');
        C = get(hGrid, 'Cols');

840 % The ROI array should only have names and vectors of numeric values
        % describing the rectangular cropping regions.
        cvROIs = cell(R-1, 2);

        for r = 1:R-1,
845     for c = 0:C-1,
            set(hGrid, 'row', r, 'col', c);
            sTxt = get(hGrid, 'text');

            % if the text is digit, wspace, or punct evaluate as numeric
850     bIsWsPunct = isstrprop(sTxt, 'wspace') | isstrprop(sTxt, 'punct');
            bIsNum = all(isstrprop(sTxt, 'digit') | bIsWsPunct);

```

```

%         printf('%12s%3d', sTxt, bIsNum)

855     if c == 0,
        % the only strings to return are the ROI names
        vVal = sTxt;
        cvROIs{r, c+1} = vVal;
    else
860         if bIsNum,
            vVal = str2num(sTxt);
        else
            % hopefully it's impossible to get here.
            vVal = [];
865         end;
        cvROIs{r, 2} = [cvROIs{r, 2}, vVal];
    end;
end;

end;
end;
870 set(hGrid, 'row', 1, 'col', 0);

% --- Executes on button press in pbLoadROI.
function pbLoadROI_Callback(hObject, eventdata, handles)
% hObject     handle to pbLoadROI (see GCBO)
875 % eventdata reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% User selection of ROI defining file.
[fname, fpath] = uigetfile('*.mat', 'Select ROI File');
880 if fname ~= 0,
    sFullPath = [fpath, filesep, fname];
    csNewNames = who('-file', sFullPath);
    stNewROIs = load(sFullPath, csNewNames{:});

885     cvNewROIs = cell(length(stNewROIs), 2);
    for i = 1:length(stNewROIs),
        cvNewROIs(i, :) = {csNewNames{i}, stNewROIs.(csNewNames{i})};
    end;

890     handles.rois = imsqt_roiaddreplace(handles.rois, cvNewROIs);
    imsqt_roigridupdate(handles)
end;

guidata(hObject, handles);
895

% -----
function imsqt_roigridupdate(handles)
cvROIs = handles.rois;
900 cvROIGrid = cell(size(cvROIs, 1), 5);
cvROIGrid(:, 1) = cvROIs(:, 1);
for i = 1:size(cvROIGrid, 1),
    roicell = num2cell(cvROIs{i, 2});
    if isempty(roicell),
905         roicell = {'auto', 'auto', 'auto', 'auto'};
    end;
    cvROIGrid(i, 2:end) = roicell;
end;
end;

```

```

910 imsqt_fillgrid(handles.roiGrid, handles.roiGridCols, cvROIGrid);

% -----
function cvCurROIs = imsqt_roiaddreplace(cvCurROIs, cvNewROIs)
915 csNewNames = cvNewROIs(:, 1);
    csCurNames = cvCurROIs(:, 1);

% search for new names that match cur names, replace corresponding data
    idx = find(ismember(csNewNames, csCurNames));
920 for i = 1:length(idx),
        cvCurROIs{...
            strmatch(csNewNames{idx(i)}, csCurNames, 'exact'), ...
                2} = cvNewROIs{idx(i), 2};
    end;
925
% search for new names that don't match cur names and append to list
    idx = find(~ismember(csNewNames, csCurNames));
    for i = 1:length(idx),
        cvCurROIs(end+1, :) = cvNewROIs(idx(i), :);
930 end;

function edtMaxDisp_Callback(hObject, eventdata, handles)
935 % hObject    handle to edtMaxDisp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtMaxDisp as text
940 %         str2double(get(hObject,'String')) returns contents of edtMaxDisp
%         as a double

% --- Executes during object creation, after setting all properties.
945 function edtMaxDisp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtMaxDisp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

950 % Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
955 end

% --- Executes on button press in chkVerboseTrk.
function chkVerboseTrk_Callback(hObject, eventdata, handles)
960 % hObject    handle to chkVerboseTrk (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of chkVerboseTrk
965

```

```

function edtTrkMem_Callback(hObject, eventdata, handles)
% hObject    handle to edtTrkMem (see GCBO)
970 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtTrkMem as text
%         str2double(get(hObject,'String')) returns contents of edtTrkMem as
975 %         a double
handles.track.mem = str2double(get(hObject, 'string'));
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
980 function edtTrkMem_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtTrkMem (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

985 % Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
                  get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
990 end

% --- Executes on button press in chkQntBlob.
function chkQntBlob_Callback(hObject, eventdata, handles)
995 % hObject    handle to chkQntBlob (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of chkQntBlob
1000 handles.quant.blobout = get(hObject, 'value');
    guidata(hObject, handles);

1005 % --- Executes on button press in chkSegDisp.
function chkSegDisp_Callback(hObject, eventdata, handles)
% hObject    handle to chkSegDisp (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1010
% Hint: get(hObject,'Value') returns toggle state of chkSegDisp
handles.segment.display = get(hObject, 'value');
    guidata(hObject, handles);

1015
% --- Executes on selection change in pmnSegDispMethod.
function pmnSegDispMethod_Callback(hObject, eventdata, handles)
% hObject    handle to pmnSegDispMethod (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
1020 % handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns pmnSegDispMethod contents
%         as cell array contents{get(hObject,'Value')} returns selected item

```

```

%           from pmnSegDispMethod
1025 contents = get(hObject, 'string');
handles.segment.displaymethod = contents{get(hObject, 'value')};
guidata(hObject, handles);

1030

function edtTrkMinTrajLen_Callback(hObject, eventdata, handles)
% hObject    handle to edtTrkMinTrajLen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1035 % handles  structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtTrkMinTrajLen as text
%        str2double(get(hObject,'String')) returns contents of
%        edtTrkMinTrajLen as a double
1040 handles.track.minlength = str2double(get(hObject, 'string'));
guidata(hObject, handles);

1045

function edtTrkMaxDisp_Callback(hObject, eventdata, handles)
% hObject    handle to edtTrkMaxDisp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1050

% Hints: get(hObject,'String') returns contents of edtTrkMaxDisp as text
%        str2double(get(hObject,'String')) returns contents of
%        edtTrkMaxDisp as a double
handles.track.maxdisp = str2double(get(hObject, 'string'));
1055 guidata(hObject, handles);

% --- Executes on button press in chkTrkVerbose.
1060 function chkTrkVerbose_Callback(hObject, eventdata, handles)
% hObject    handle to chkTrkVerbose (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

1065 % Hint: get(hObject,'Value') returns toggle state of chkTrkVerbose
handles.track.verbose = get(hObject, 'value');
guidata(hObject, handles);

% -----
1070 function handles = imsgt_collectinfo(handles)
handles.info.path      = get(handles.edtExptPath, 'string');
handles.info.corrpath  = get(handles.edtCorrPath, 'string');
% handles.info.prefix  = get(handles.edtExptPrefix, 'string');
% handles.info.id      = get(handles.edtExptID, 'string');
1075 handles.info.locid   = str2double(get(handles.edtLocID, 'string'));

% iIdxStart = str2num(get(handles.edtIdxStart, 'string'));
% iIdxStep  = str2num(get(handles.edtIdxStep, 'string'));
% iIdxStop  = str2num(get(handles.edtIdxStop, 'string'));
1080 % handles.info.indices = [iIdxStart:iIdxStep:iIdxStop];

```

```

handles.info.indices = get(handles.edtIndices, 'value');
handles.info.indicesstr = get(handles.edtIndices, 'string');

1085 % --- Executes on button press in pbSaveROI.
function pbSaveROI_Callback(hObject, eventdata, handles)
% hObject    handle to pbSaveROI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1090
sExptPath = imsq_getexptpath(handles);
if ~isempty(sExptPath),
    cvROIs = imsq_parseroigrid(handles.roigrid);
    for i = 1:size(cvROIs, 1),
1095         sROI = cvROIs{i, 1};
            nROI = cvROIs{i, 2};
            eval([sROI ' = [' num2str(nROI) '];']);
            sFileName = sprintf('%s%s%.03d.mat', sExptPath, ...
                                filesep, ...
1100                                sROI, ...
                                handles.info.locid);

            save(sFileName, sROI);
        end;

1105 else
    errordlg('Experiment path information not specified!', 'Error');
end;

% --- Executes on button press in chkSegOverwrite.
1110 function chkSegOverwrite_Callback(hObject, eventdata, handles)
% hObject    handle to chkSegOverwrite (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

1115 % Hint: get(hObject,'Value') returns toggle state of chkSegOverwrite
handles.segment.overwrite = get(hObject, 'value');
guidata(hObject, handles);

1120
% --- Executes on button press in chkQntOverwrite.
function chkQntOverwrite_Callback(hObject, eventdata, handles)
% hObject    handle to chkQntOverwrite (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1125 % handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of chkQntOverwrite
handles.quant.overwrite = get(hObject, 'value');
guidata(hObject, handles);
1130

% --- Executes on button press in chkTrkOverwrite.
function chkTrkOverwrite_Callback(hObject, eventdata, handles)
1135 % hObject    handle to chkTrkOverwrite (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

% Hint: get(hObject,'Value') returns toggle state of chkTrkOverwrite
1140 handles.track.overwrite = get(hObject, 'value');
    guidata(hObject, handles);

1145 % -----
function MenuFile_Callback(hObject, eventdata, handles)
% hObject    handle to MenuFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1150

% -----
function imsqt_updateinfo(handles)
set(handles.edtExptPath, 'string', handles.info.path);
1155 set(handles.edtCorrPath, 'string', handles.info.corrpath);
set(handles.edtLocID, 'string', num2str(handles.info.locid));

if ~isfield(handles.info, 'indicesstr'),
1160     iIdxStep = unique(diff(handles.info.indices));
    if isempty(iIdxStep), iIdxStep = 1; end;
    if length(iIdxStep) == 1,
        iIdxStart = min(handles.info.indices);
        iIdxStop = max(handles.info.indices);
1165     handles.info.indicesstr = sprintf('%d:%d:%d', iIdxStart, ...
                                           iIdxStep, ...
                                           iIdxStop);
    else
        handles.info.indicesstr = num2str(handles.info.indices);
1170     end;
end;

set(handles.edtIndices, 'string', handles.info.indicesstr);
set(handles.edtIndices, 'value', handles.info.indices);
1175
% set(handles.edtIdxStart, 'string', num2str(iIdxStart));
% set(handles.edtIdxStep, 'string', num2str(iIdxStep));
% set(handles.edtIdxStop, 'string', num2str(iIdxStop));

1180
% -----
function imsqt_updatesegment(handles)
set(handles.edtSegDataPath, 'string', handles.segment.path);
imsqt_setpnmvalbystr(handles.pmnSegSegmenter, handles.segment.algorithm);
1185 set(handles.chkSegDisp, 'value', handles.segment.display);
imsqt_setpnmvalbystr(handles.pmnSegDispMethod, ...
                    handles.segment.displaymethod);
set(handles.chkSegOverwrite, 'value', handles.segment.overwrite);

1190 % -----
function imsqt_updatequant(handles)
set(handles.edtObjDataPath, 'string', handles.quant.path);
set(handles.chkQntVerbose, 'value', handles.quant.verbose);
set(handles.chkQntBlob, 'value', handles.quant.blobout);

```

```

1195 set(handles.chkQntOverwrite, 'value', handles.quant.overwrite);

% -----
function imsqt_updatetrack(handles)
1200 set(handles.edtTrjDatapath, 'string', handles.track.path);
    set(handles.edtTrkMinTrajLen, 'string', num2str(handles.track.minlength));
    set(handles.edtTrkMaxDisp, 'string', num2str(handles.track.maxdisp));
    set(handles.edtTrkMem, 'string', num2str(handles.track.mem));
    set(handles.chkTrkVerbose, 'value', handles.track.verbose);
1205 set(handles.chkTrkOverwrite, 'value', handles.track.overwrite);

% -----
function imsqt_setpmnvalbystr(hpmn, strval)
1210 set(hpmn, 'value', strmatch(strval, get(hpmn, 'string'), 'exact'));

% -----
function MenuFileLoad_Callback(hObject, eventdata, handles)
1215 % hObject    handle to MenuFileLoad (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
[fname fpath] = uigetfile('*.mat', 'Retrieve Run-Time Data', ...
    [handles.uidata.appname 'data.mat']);
1220 if fname ~= 0,
    set(handles.figure1, 'name', [handles.uidata.appname ' - ' fname]);
    handles.uidata.cwd = fpath;
    handles.uidata.file = [fpath filesep fname];
    cd(fpath);
1225
    load(handles.uidata.file, 'data');

    handles.info = data.info;
    imsqt_updateinfo(handles);
1230
    handles.channels = data.channels;
    imsqt_fillgrid(handles.channelgrid, ...
        handles.channelgridcols, ...
        handles.channels);
1235
    handles.rois = data.rois;
    imsqt_roigridupdate(handles);

    % segmentation panel defaults
1240 handles.segment = data.segment;
    imsqt_updatesegment(handles);

    % quantification panel defaults
    handles.quant = data.quant;
1245 imsqt_updatequant(handles);

    % tracking panel defaults
    handles.track = data.track;
    imsqt_updatetrack(handles);
1250 end;
    guidata(hObject, handles);

```



```

% -----
function MenuFileSave_Callback(hObject, eventdata, handles)
1255 % hObject    handle to MenuFileSave (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% update fields that may have changed that do not have callbacks.
1260 handles = imsqc_collectinfo(handles);
handles.rois = imsqc_parseroigrid(handles.roigrid);
handles.channels = imsqc_parsechannelgrid(handles);

data.info = handles.info;
1265 data.rois = handles.rois;
data.channels = handles.channels;
data.segment = handles.segment;
data.quant = handles.quant;
data.track = handles.track;
1270
if isempty(handles.uidata.file),
    [fname fpath] = uiputfile('*.mat', 'Save Run-Time Data', ...
                             [handles.uidata.appname 'data.mat']);
    if fname ~= 0,
1275        set(handles.figure1, 'name', [handles.uidata.appname ' - ' fname]);
        handles.uidata.cwd = fpath;
        handles.uidata.file = [fpath filesep fname];
        cd(fpath);
        save(handles.uidata.file, 'data');
1280    end;
    else
        save(handles.uidata.file, 'data');
    end;
guidata(hObject, handles);
1285
% -----
function MenuFileSaveAs_Callback(hObject, eventdata, handles)
% hObject    handle to MenuFileSaveAs (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
1290 % handles      structure with handles and user data (see GUIDATA)

% update fields that may have changed that do not have callbacks.
handles = imsqc_collectinfo(handles);
handles.rois = imsqc_parseroigrid(handles.roigrid);
1295 handles.channels = imsqc_parsechannelgrid(handles);

data.info = handles.info;
data.rois = handles.rois;
data.channels = handles.channels;
1300 data.segment = handles.segment;
data.quant = handles.quant;
data.track = handles.track;

[fname fpath] = uiputfile('*.mat', 'Save Run-Time Data', ...
1305                             [handles.uidata.appname 'data.mat']);
if fname ~= 0,
    set(handles.figure1, 'name', [handles.uidata.appname ' - ' fname]);
    handles.uidata.cwd = fpath;

```

```

        handles.uidata.file = [fpath filesep fname];
1310    cd(fpath);
        save(handles.uidata.file, 'data');
    end;
    guidata(hObject, handles);

1315 % -----
    function MenuFileQuit_Callback(hObject, eventdata, handles)
    % hObject    handle to MenuFileQuit (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
1320
    closereq

    function edtCorrPath_Callback(hObject, eventdata, handles)
1325 % hObject    handle to edtCorrPath (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of edtCorrPath as text
1330 %         str2double(get(hObject,'String')) returns contents of edtCorrPath
    %         as a double
    handles.info.corrpath = get(hObject, 'string');
    guidata(hObject, handles);

1335 % --- Executes during object creation, after setting all properties.
    function edtCorrPath_CreateFcn(hObject, eventdata, handles)
    % hObject    handle to edtCorrPath (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called
1340
    % Hint: edit controls usually have a white background on Windows.
    %         See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
1345    set(hObject,'BackgroundColor','white');
    end

    % --- Executes on button press in pbCorrPathBrowse.
1350 function pbCorrPathBrowse_Callback(hObject, eventdata, handles)
    % hObject    handle to pbCorrPathBrowse (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    sDir = uigetdir;
1355 if sDir ~= 0,
        set(handles.edtCorrPath, 'string', sDir);
    end;

1360
    % -----
    function MenuTools_Callback(hObject, eventdata, handles)
    % hObject    handle to MenuTools (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
1365 % handles    structure with handles and user data (see GUIDATA)

```

```

% -----
function MenuToolsSegEdit_Callback(hObject, eventdata, handles)
1370 % hObject    handle to MenuToolsSegEdit (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
segedit('file', [handles.info.path, ...
                handles.segment.path, ...
1375                filesep, ...
                sprintf('%03d%c%04d.mat', handles.info.locid, ...
                        filesep, ...
                        handles.info.indices(1))...
                ]);
1380

function edtIndices_Callback(hObject, eventdata, handles)
1385 % hObject    handle to edtIndices (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtIndices as text
1390 %          str2double(get(hObject,'String')) returns contents of edtIndices
%          as a double

set(hObject, 'Value', sort(str2num(get(hObject, 'String'))));

1395 % --- Executes during object creation, after setting all properties.
function edtIndices_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtIndices (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called
1400

% Hint: edit controls usually have a white background on Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
                  get(0,'defaultUicontrolBackgroundColor'))
1405     set(hObject,'BackgroundColor','white');
end

1410
% -----
function MenuToolsTrackEdit_Callback(hObject, eventdata, handles)
% hObject    handle to MenuToolsTrackEdit (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
1415 % handles      structure with handles and user data (see GUIDATA)
if ~isempty(handles.uidata.file)
    if exist([handles.info.path, handles.track.path], 'dir'),
        trackedit('file', [handles.info.path, ...
                            filesep, ...
1420                            handles.uidata.file]);
    else
        errordlg('Cannot find tracking data', 'Tracking Data Not Found');

```

```
        end;
    else
1425     errordlg(['...
            'Cannot find IMSQT info file.' ...
            ' Make sure that it is saved in the path specified by '...
            ''Experiment Path'','], 'File Not Found');
    end;
1430
```

B.7 Complete source code for SEGEDIT

```

function varargout = segedit(varargin)
% SEGEDIT M-file for segedit.fig
%   SEGEDIT, by itself, creates a new SEGEDIT or raises the existing
%   singleton*.
5 %
%   H = SEGEDIT returns the handle to a new SEGEDIT or the handle to
%   the existing singleton*.
%
%   SEGEDIT('CALLBACK',hObject,eventData,handles,...) calls the local
10 %   function named CALLBACK in SEGEDIT.M with the given input arguments.
%
%   SEGEDIT('Property','Value',...) creates a new SEGEDIT or raises the
%   existing singleton*. Starting from the left, property value pairs
%   are applied to the GUI before segedit_OpeningFunction gets called.
15 %   An unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to segedit_OpeningFcn via
%   varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
20 %   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.
25 % Edit the above text to modify the response to help segedit

% Last Modified by GUIDE v2.5 04-Apr-2006 15:15:57

30 % Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @segedit_OpeningFcn, ...
35                  'gui_OutputFcn', @segedit_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
40 end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
45    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

50 % --- Executes just before segedit is made visible.
function segedit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB

```

```

55 % handles      structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to segedit (see VARARGIN)

    % Choose default command line output for segedit
handles.output = hObject;

60
    % set default property values
pos = get(handles.figure1, 'position');
handles.uiprops.minfigsize = pos(3:4);

65 handles.uiprops.appname = get(hObject, 'name');
handles.uiprops.color.cur = [1 0 0];
handles.uiprops.color.preview = [1 1 0];
handles.uiprops.color.add = [0 1 0];
handles.uiprops.color.addglobal = [1 0 1];
70 handles.uiprops.color.erase = [0 0 1];
handles.uiprops.color.eraseglobal = [0 1 1];
handles.uiprops.linewidth.area = 1;
handles.uiprops.linewidth.line = 3;
handles.issaved = false;

75
set(handles.tbDisp, 'String', 'Mask On');
set(handles.figure1, 'KeyPressFcn', @segedit_keypress);

handles.maskglobal.add = [];
80 handles.maskglobal.erase = [];

    % collect original object positions for application resize
hUI = findobj(handles.figure1, ...
    '-depth', 1, ...
85     'type', 'uicontrol', ...
    '-or', 'type', 'uipanel', ...
    '-or', 'type', 'axes');
hUI = [handles.figure1; hUI];
set(hUI, 'units', 'pixels');
90 handles.uisize = get(handles.figure1, 'position');

    % set the default state of the application. This is how the app will start
    % if run by itself with no command line arguments

95 % process command line options
opts = getopt(varargin);

handles.file      = parseopts('file', opts, []);
handles.cwd      = cd;
100 %setWindowState(handles.figure1, parseopts('windowstate', opts, 'restore'));

    if ~isempty(handles.file),
        [fpath, fname, fext, fver] = fileparts(handles.file);

105     if isempty(fpath), fpath = cd; end;
        fname = [fname fext];

        cd(fpath);
handles = segedit_loadfile(handles, fpath, fname);
110 segedit_load(handles); % enable the gui for user input
segedit_disp(handles); % display the segmentation image and mask

```

```

        segedit_browseinit(handles); % initialize the browsing slider
    else,
        handles.seg          = [];
115     handles.image         = [];
        handles.maskall.orig = [];
        handles.maskall.curr = handles.maskall.orig;
        handles.maskqnt = handles.maskall;

120     set(handles.axes1, 'Visible', 'off');
        segedit_enable(handles, 'off');

    end;

125 % Update handles structure
    guidata(hObject, handles);

    % UIWAIT makes segedit wait for user response (see UIRESUME)
    % uiwait(handles.figure1);
130

    % --- Outputs from this function are returned to the command line.
    function varargout = segedit_OutputFcn(hObject, eventdata, handles)
    % varargout cell array for returning output args (see VARARGOUT);
135 % hObject    handle to figure
    % eventdata reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Get default command line output from handles structure
140 varargout{1} = handles.output;
    % varargout{1} = handles.seg;

    % delete(handles.figure1);

145
    % --- Executes on button press in pbAdd.
    function pbAdd_Callback(hObject, eventdata, handles)
    % hObject    handle to pbAdd (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
150 % handles    structure with handles and user data (see GUIDATA)
    statmsg = 'Add Poly Region';
    updatestatus(handles, statmsg);

    hax = handles.axes1;
155 axes(hax);
    bnd = bwboundaries(roipoly);
    if ~ishold(hax), hold on, end;
    hroi = plot(bnd{1}(:, 2), bnd{1}(:, 1), ...
                'linestyle', '-', ...
160                'color', handles.uiprops.color.add, ...
                'linewidth', handles.uiprops.linewidth.area);
    set(hroi, 'hitTest', 'on', 'ButtonDownFcn', @segedit_onclick);

    updatestatus(handles, '');
165 drawnow

    guidata(hObject, handles);

```

```

% --- Executes on button press in pbErase.
170 function pbErase_Callback(hObject, eventdata, handles)
% hObject    handle to pbErase (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
sEraseType = get(handles.pmnEraseType, 'String');
175 sEraseType = sEraseType{get(handles.pmnEraseType, 'Value')};
    bGlobal = get(handles.chkMaskApplyAll, 'Value');

    hax = handles.axes1;
    axes(hax);
180
    if strcmpi(sEraseType, 'area'),
        statmsg = 'Draw Erasing Region';
        iColor = handles.uiprops.color.erase;
        if bGlobal,
185             statmsg = 'Draw Global Erasing Region';
                iColor = handles.uiprops.color.eraseglobal;
        end;
        updatestatus(handles, statmsg);

190     msk = roipoly;
        bnd = bwboundaries(msk);
        if ~ishold(hax), hold on, end;
        heroi = plot(bnd{1}(:, 2), bnd{1}(:, 1), ...
                    'linestyle', '-', ...
195                    'color', iColor, ...
                    'linewidth', handles.uiprops.linewidth.area);
        set(heroi, 'hitest', 'on', 'buttondownfcn', @segedit_onclick);

        if bGlobal,
200             if isempty(handles.maskglobal.erase),
                    handles.maskglobal.erase = msk;
                else,
                    handles.maskglobal.erase = handles.maskglobal.erase | msk;
                end;
205     end;

    else,
        % default to erasing lines
        statmsg = 'Draw Erasing Line';
210     updatestatus(handles, statmsg);

        [x, y] = getline(hax);
        if ~ishold(hax), hold on, end;
        heline = plot(x, y, ...
215                    'color', handles.uiprops.color.erase, ...
                    'linewidth', handles.uiprops.linewidth.line);
        set(heline, 'hitest', 'on', 'buttondownfcn', @segedit_onclick);

    end;
220
    updatestatus(handles, '');
    drawnow

    guidata(hObject, handles);
225

```



```

% --- Executes on button press in pbUpdate.
function pbUpdate_Callback(hObject, eventdata, handles)
% hObject    handle to pbUpdate (see GCBO)
230 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(hObject, 'Enable', 'Off');
handles = segedit_update(handles);
set(hObject, 'Enable', 'on');
235 drawnow
    guidata(hObject, handles);

% -----
240 function MenuItemOpen_Callback(hObject, eventdata, handles)
% hObject    handle to MenuItemOpen (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[fname fpath]= uigetfile('*.mat');
245 if ~isequal(fname, 0)
    cd(fpath);
    handles = segedit_loadfile(handles, fpath, fname);
    segedit_load(handles); % enable the gui for user input
    segedit_disp(handles); % display the segmentation image and mask
250    segedit_browseinit(handles); % initialize the browsing slider
end
    guidata(hObject, handles);

255 % -----
function MenuItemQuit_Callback(hObject, eventdata, handles)
% hObject    handle to MenuItemQuit (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
260 guidata(hObject, handles);

delete(handles.figure1)
% uiresume(handles.figure1);

265 % -----
function MenuFile_Callback(hObject, eventdata, handles)
% hObject    handle to MenuFile (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
270

275 function edtChSeg_Callback(hObject, eventdata, handles)
% hObject    handle to edtChSeg (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

280 % Hints: get(hObject,'String') returns contents of edtChSeg as text
%         str2double(get(hObject,'String')) returns contents of edtChSeg as
%         a double

```

```

285 % --- Executes during object creation, after setting all properties.
    function edtChSeg_CreateFcn(hObject, eventdata, handles)
        % hObject    handle to edtChSeg (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    empty - handles not created until after all CreateFcns called
290
        % Hint: edit controls usually have a white background on Windows.
        %         See ISPC and COMPUTER.
        if ispc
            set(hObject,'BackgroundColor','white');
295 else
            set(hObject,'BackgroundColor',...
                get(0,'defaultUicontrolBackgroundColor'));
        end

300
        % -----
        function edit2_Callback(hObject, eventdata, handles)
            % hObject    handle to edit2 (see GCBO)
            % eventdata  reserved - to be defined in a future version of MATLAB
305 % handles    structure with handles and user data (see GUIDATA)

            % Hints: get(hObject,'String') returns contents of edit2 as text
            %         str2double(get(hObject,'String')) returns contents of edit2 as a
            %         double
310

            % --- Executes during object creation, after setting all properties.
            function edit2_CreateFcn(hObject, eventdata, handles)
                % hObject    handle to edit2 (see GCBO)
315 % eventdata  reserved - to be defined in a future version of MATLAB
                % handles    empty - handles not created until after all CreateFcns called

                % Hint: edit controls usually have a white background on Windows.
                %         See ISPC and COMPUTER.
320 if ispc
                    set(hObject,'BackgroundColor','white');
                else
                    set(hObject,'BackgroundColor',...
                        get(0,'defaultUicontrolBackgroundColor'));
325 end

330 % --- Executes during object creation, after setting all properties.
    function edtStatus_CreateFcn(hObject, eventdata, handles)
        % hObject    handle to edtStatus (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    empty - handles not created until after all CreateFcns called
335

        % -----
        function edtStatus_Callback(hObject, eventdata, handles)
            % hObject    handle to edtStatus (see GCBO)

```

```

340 % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'String') returns contents of edtStatus as text
    %         str2double(get(hObject,'String')) returns contents of edtStatus as
345 %         a double

% -----
function segedit_enable(handles, sState)
350 hObjs = findobj('type', 'uicontrol', '-and', {...
    'style', 'pushbutton', ...
    '-or', 'style', 'pushbutton', ...
    '-or', 'style', 'togglebutton', ...
    '-or', 'style', 'popupmenu', ...
355    '-or', 'style', 'edit', ...
    }, ...
    '-and', '-not', 'tag', 'edtStatus' ...
    );
set(hObjs, 'enable', sState);
360

% -----
function segedit_disp(handles)
hold off;
segedit_disp_image(handles);
365 hold on;
segedit_disp_mask(handles);

% -----
function segedit_disp_image(handles)
370 axes(handles.axes1);
him = imshow(handles.image, [], 'InitialMagnification', 'fit');
set(him, 'hitest', 'off');

% -----
375 function segedit_disp_mask(handles)
axes(handles.axes1);
BND = bwboundaries(handles.maskqnt.curr, 'noholes');
for i = 1:length(BND),
    plot(BND{i}(:, 2), BND{i}(:,1), ...
380         'color', handles.uiprops.color.cur, ...
         'linestyle', '-', ...
         'linewidth', handles.uiprops.linewidth.area)
    hold on
end;
385
if ~isempty(handles.maskglobal.erase) && sum(sum(handles.maskglobal.erase)) > 0,
    BND = bwboundaries(handles.maskglobal.erase, 'noholes');
    for i = 1:length(BND),
        plot(BND{i}(:, 2), BND{i}(:,1), ...
390             'color', handles.uiprops.color.eraseglobal, ...
             'linestyle', '-', ...
             'linewidth', handles.uiprops.linewidth.area)
        hold on
    end;
395 end;

```

```

if ~isempty(handles.maskglobal.add) && sum(sum(handles.maskglobal.add)) > 0,
    BND = bwboundaries(handles.maskglobal.add, 'noholes');
    for i = 1:length(BND),
400         plot(BND{i}(:, 2), BND{i}(:,1), ...
                'color', handles.uiprops.color.addglobal, ...
                'linestyle', '-', ...
                'linewidth', handles.uiprops.linewidth.area)
            hold on
405     end;
    end;

    BND = bwboundaries(handles.maskall.curr & ~handles.maskqnt.curr, 'noholes');
    for i = 1:length(BND),
410         plot(BND{i}(:, 2), BND{i}(:,1), ...
                'color', handles.uiprops.color.cur, ...
                'linestyle', ':', ...
                'linewidth', handles.uiprops.linewidth.area)
            hold on
415     end;
    hold off
    axis off ij, axis([0 handles.seg.imsz(2) 0 handles.seg.imsz(1)])
    set(findobj('type', 'line'), ...
        'hitest', 'on', ...
420         'buttondownfcn', @segedit_onclick);

    % -----
    function updatestatus(handles, statusmsg)
    set(handles.edtStatus, 'String', statusmsg);
425
    % -----
    function I = segedit_getimg(handles, varargin)
    I = getimg( ...
        handles.seg.channel, ...
430         handles.seg.locid, ...
        handles.seg.imgid, ...
        'basepath', [handles.cwd, '\..\..\']) - 2^15;
    I = imcrop(I, handles.seg.roi);

435

    % -----
    function MenuItemSave_Callback(hObject, eventdata, handles)
    % hObject    handle to MenuItemSave (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
440 % handles    structure with handles and user data (see GUIDATA)

    % write out seg data to a .mat file pull in values that were not edited by
    % this application, overwrite those that were.
445
    seg = handles.seg;

    seg.objqntidx = find(bwperim(handles.maskqnt.curr));
    seg.objallidx = find(bwperim(handles.maskall.curr));
450
    handles.seg = seg;
    save(sprintf('%s%s%s', handles.cwd, filesep, handles.file), 'seg');

```

```

handles.maskall.orig = handles.maskall.curr;
455 handles.maskqnt.orig = handles.maskqnt.curr;

checkSaved(handles);

guidata(hObject, handles);
460

% -----
function MenuItemSaveAs_Callback(hObject, eventdata, handles)
465 % hObject    handle to MenuItemSaveAs (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% write out seg data to a .mat file that the user specifies
470 seg = handles.seg;

seg.objqntidx = find(bwperim(handles.maskqnt.curr));
seg.objallidx = find(bwperim(handles.maskall.curr));

475 [fname fpath] = uiputfile('*.mat', 'Save As', handles.file);
if ~isequal(fname, 0),
    handles.seg = seg;
    save(sprintf('%s/%s', fpath, fname), 'seg');
    set(gcf, 'name', sprintf('%s - %s', handles.uiprops.appname, fname));
480
    handles.maskall.orig = handles.maskall.curr;
    handles.maskqnt.orig = handles.maskqnt.curr;

    handles.issaved = true;
485 end;
guidata(hObject, handles);

function checkSaved(handles)
490 % determine if the segmentation needs to be saved by comparing
% the original mask to the current mask
handles.issaved = ...
    (length(find(handles.maskall.curr ~= handles.maskall.orig)) == 0);

495 if ~handles.issaved,
    set(gcf, 'name', sprintf('%s - %s*', handles.uiprops.appname, ...
        handles.file));
    else,
    set(gcf, 'name', sprintf('%s - %s', handles.uiprops.appname, ...
500        handles.file));
    end;

guidata(gcf, handles);

505
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
510

```

```

% Hints: get(hObject,'String') returns contents of edit4 as text
%         str2double(get(hObject,'String')) returns contents of edit4 as a
%         double

515 % --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
520 % handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc
525     set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',...
        get(0,'defaultUicontrolBackgroundColor'));
end

530 % -----
function segedit_load(handles);
set(handles.MenuItemSave, 'Enable', 'on');
set(handles.MenuItemSaveAs, 'Enable', 'on');
535 set(handles.edtChSeg, 'String', handles.seg.channel);
set(handles.edit2, 'String', handles.seg.locid);
set(handles.edit4, 'String', handles.seg.imgid);
segedit_enable(handles, 'on');

540 % -----
function MenuItemReturn_Callback(hObject, eventdata, handles)
% hObject    handle to MenuItemReturn (see GCBO)
545 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get the current segmentation masks into the returned variable handles.seg
% and exit.
550 seg = handles.seg;

    seg.objqntidx = find(bwperim(handles.maskqnt.curr));
    seg.objallidx = find(bwperim(handles.maskall.curr));

555 handles.seg = seg;

    guidata(hObject, handles);
    uiresume(handles.figure1);

560 % -----
function MenuEdit_Callback(hObject, eventdata, handles)
% hObject    handle to MenuEdit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
565 % handles    structure with handles and user data (see GUIDATA)

```

```

% -----
function MenuItemAdd_Callback(hObject, eventdata, handles)
570 % hObject    handle to MenuItemAdd (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

575 % -----
function MenuItemErase_Callback(hObject, eventdata, handles)
% hObject    handle to MenuItemErase (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)
580

% -----
function MenuItemUpdate_Callback(hObject, eventdata, handles)
% hObject    handle to MenuItemUpdate (see GCBO)
585 % eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% -----
590 function MenuItemDelete_Callback(hObject, eventdata, handles)
% hObject    handle to MenuItemDelete (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

595

% --- Executes on selection change in pmnMorphOp.
function pmnMorphOp_Callback(hObject, eventdata, handles)
600 % hObject    handle to pmnMorphOp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns pmnMorphOp contents as
605 %         cell array contents{get(hObject,'Value')} returns selected item
%         from pmnMorphOp

contents = get(hObject, 'String');
handles.morph.op = lower(contents{get(hObject, 'Value')});
610 guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
615 function pmnMorphOp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pmnMorphOp (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      empty - handles not created until after all CreateFcns called

620 % Hint: popupmenu controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), ...
                 get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');

```

```

625 end

    handles.morph.op = 'dilate';
    guidata(hObject, handles);

630 % --- Executes on selection change in pmnStrelType.
    function pmnStrelType_Callback(hObject, eventdata, handles)
        % hObject    handle to pmnStrelType (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    structure with handles and user data (see GUIDATA)
635
        % Hints: contents = get(hObject,'String') returns pmnStrelType contents as
        %         cell array contents{get(hObject,'Value')} returns selected item
        %         from pmnStrelType

640 contents = get(hObject, 'String');
        handles.morph.strel.type = lower(contents{get(hObject, 'Value')});

        guidata(hObject, handles);

645 % --- Executes during object creation, after setting all properties.
    function pmnStrelType_CreateFcn(hObject, eventdata, handles)
        % hObject    handle to pmnStrelType (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    empty - handles not created until after all CreateFcns called
650
        % Hint: popupmenu controls usually have a white background on Windows.
        %         See ISPC and COMPUTER.
        if ispc && isequal(get(hObject,'BackgroundColor'), ...
            get(0,'defaultUicontrolBackgroundColor'))
655     set(hObject,'BackgroundColor','white');
        end

        handles.morph.strel.type = 'disk';
        guidata(hObject, handles);
660

        function edtStrelSize_Callback(hObject, eventdata, handles)
            % hObject    handle to edtStrelSize (see GCBO)
            % eventdata  reserved - to be defined in a future version of MATLAB
665 % handles    structure with handles and user data (see GUIDATA)

            % Hints: get(hObject,'String') returns contents of edtStrelSize as text
            %         str2double(get(hObject,'String')) returns contents of edtStrelSize
            %         as a double
670
            handles.morph.strel.size = str2num(get(hObject, 'String'));
            guidata(hObject, handles);

            % --- Executes during object creation, after setting all properties.
675 function edtStrelSize_CreateFcn(hObject, eventdata, handles)
                % hObject    handle to edtStrelSize (see GCBO)
                % eventdata  reserved - to be defined in a future version of MATLAB
                % handles    empty - handles not created until after all CreateFcns called

680 % Hint: edit controls usually have a white background on Windows.
                %         See ISPC and COMPUTER.

```



```

        if ispc && isequal(get(hObject,'BackgroundColor'), ...
            get(0,'defaultUiControlBackgroundColor'))
            set(hObject,'BackgroundColor','white');
685 end

        handles.morph.strel.size = 3;
        guidata(hObject, handles);

690 % --- Executes on button press in pbMorph.
        function pbMorph_Callback(hObject, eventdata, handles)
            % hObject    handle to pbMorph (see GCBO)
            % eventdata  reserved - to be defined in a future version of MATLAB
            % handles    structure with handles and user data (see GUIDATA)
695
            bApplyAll = get(handles.chkMorphApplyAll, 'value');

            if bApplyAll,
                % Display a warning notifying the user that this operation updates all
700                % files in the current working directory without an undo
                button = questdlg(['...
                    'This will irreversibly modify all files in the current' ...
                    'working directory. Do you wish to continue?'],...
                    'Morph All','Yes','No','No');
705                if strcmpi(button, 'yes'),
                    % this could take a while.  display a waitbar to update progress
                    hProg = waitbar(0, 'Processing Global Morphological Operation');
                    nFiles = length(handles.files);
                    tLoopElaps = [];

710                    for i = 1:nFiles,
                        tLoopInit = clock;
                        updateprogress(hProg, i, nFiles, tLoopElaps);

715                        waitbar(i/nFiles, hProg, ...
                            'Processing Global Morphological Operation');
                        fpath = [handles.cwd filesep handles.files{i}];

                        load(fpath, 'seg');
720                        msk = bndidx2mask(seg.imsiize, seg.objallidx);
                        msk = segedit_morphobjs(handles, msk);

                        seg.objallidx = find(bwperim(msk));
                        seg.objqntidx = find(bwperim(imclearborder(msk)));
725                        save(fpath, 'seg');

                        tLoopElaps = [tLoopElaps etime(clock, tLoopInit)];
                    end;
730                    close(hProg);

                    % refresh the currently displayed file
                    handles= segedit_loadfile(handles, handles.cwd, handles.file);
                    segedit_load(handles);
735                    segedit_disp(handles);
                    segedit_browseinit(handles);
            end;

```

```

else,
740     handles.maskall.curr = segedit_morphobjs(handles, ...
                                           handles.maskall.curr);
        handles.maskqnt.curr = imclearborder(handles.maskall.curr);
        segedit_disp(handles);

745     checkSaved(handles);
        drawnow
    end;

    guidata(hObject, handles);
750

    % -----
    function msk = segedit_morphobjs(handles, msk)
    switch handles.morph.strel.type
755     case 'disk',
            se = strel('disk', handles.morph.strel.size);
        case 'square',
            se = strel('square', handles.morph.strel.size);
        otherwise,
760         se = strel('square', 3);
    end;
    switch handles.morph.op
        case 'dilate',
            msk = imdilate(msk, se);
765     case 'erode',
            msk = imerode(msk, se);
        case 'close',
            msk = imclose(msk, se);
        case 'open',
770         msk = imopen(msk, se);
        otherwise,
    end;

775 % --- Executes on slider movement.
    function sldBrowse_Callback(hObject, eventdata, handles)
    % hObject    handle to sldBrowse (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
780
    % Hints: get(hObject,'Value') returns position of slider
    %         get(hObject,'Min') and get(hObject,'Max') to determine range of
    %         slider
    nValue = round(get(hObject, 'Value'));
785 sValue = num2str(nValue);

    handles= segedit_loadfile(handles, handles.cwd, handles.files{nValue});
    segedit_load(handles);
    segedit_disp(handles);
790 segedit_browseinit(handles);

    guidata(hObject, handles);

    % --- Executes during object creation, after setting all properties.
795 function sldBrowse_CreateFcn(hObject, eventdata, handles)

```

```

% hObject    handle to sldBrowse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called

800 % Hint: slider controls usually have a light gray background.
    if isequal(get(hObject,'BackgroundColor'), ...
               get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
805

function edtBrowse_Callback(hObject, eventdata, handles)
% hObject    handle to edtBrowse (see GCBO)
810 % eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtBrowse as text
%         str2double(get(hObject,'String')) returns contents of edtBrowse as
815 %         a double
nFileID = str2num(get(hObject, 'String'));
fname = sprintf('%04d.mat', nFileID);
handles = segedit_loadfile(handles, handles.cwd, fname);
segedit_load(handles);
820 segedit_disp(handles);
    segedit_browseinit(handles);
    guidata(hObject, handles);

825 % --- Executes during object creation, after setting all properties.
function edtBrowse_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtBrowse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     empty - handles not created until after all CreateFcns called
830

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
    if ispc && isequal(get(hObject,'BackgroundColor'), ...
                       get(0,'defaultUicontrolBackgroundColor'))
835        set(hObject,'BackgroundColor','white');
    end

% --- Executes on button press in pbBrowsePrev.
840 function pbBrowsePrev_Callback(hObject, eventdata, handles)
% hObject    handle to pbBrowsePrev (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
nValue = round(get(handles.sldBrowse, 'Value')) - 1;
845

    if nValue >= 1,
        sValue = num2str(nValue);

        handles= segedit_loadfile(handles, handles.cwd, handles.files{nValue});
850        segedit_load(handles);
        segedit_disp(handles);
        segedit_browseinit(handles);

```

```

        guidata(hObject, handles);
855 end;

% --- Executes on button press in pbBrowseNext.
function pbBrowseNext_Callback(hObject, eventdata, handles)
% hObject    handle to pbBrowseNext (see GCBO)
860 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
nValue = round(get(handles.sldBrowse, 'Value')) + 1;

if nValue <= length(handles.files),
865     sValue = num2str(nValue);

        handles= segedit_loadfile(handles, handles.cwd, handles.files{nValue});
        segedit_load(handles);
        segedit_disp(handles);
870     segedit_browseinit(handles);

        guidata(hObject, handles);
end;

875 function files = segedit_getfiles(handles)
files = dir([handles.cwd, '\*.mat']);
files = files(find(~[files.isdir]));
files = {files.name};

880 function segedit_browseinit(handles);
nfiles = length(handles.files);
[pathstr, fname, fext] = fileparts(handles.file);

885 set(handles.sldBrowse, ...
        'Value', strmatch([fname fext], handles.files, 'exact'), ...
        'Min', 1, ...
        'Max', nfiles, ...
        'SliderStep', [1/(nfiles - 1), 10/(nfiles - 1)] ...
890 );

        set(handles.edtBrowse, 'String', str2num(fname));

895 function handles = segedit_loadfile(handles, fpath, fname)
load([fpath, '\', fname]);

        set(gcf, 'name', sprintf('%s - %s', handles.uiprops.appname, fname));
handles.seg = seg;
900 handles.cwd = fpath;
handles.file = fname;
% get the list of files in the working directory
handles.files = segedit_getfiles(handles);
handles.issaved = true;
905 handles.image = segedit_getimg(handles);

% create the masks for display from the data in the selected file
msk = bndidx2mask(handles.seg.imsizes, handles.seg.objqntidx);
handles.maskqnt.orig = msk;

```

```

910 handles.maskqnt.curr = msk;

    msk = bndidx2mask(handles.seg.imsz, handles.seg.objallidx);
    handles.maskall.orig = msk;
    handles.maskall.curr = msk;
915

    % --- Executes on button press in pbUpdateSave.
    function pbUpdateSave_Callback(hObject, eventdata, handles)
    % hObject    handle to pbUpdateSave (see GCBO)
920 % eventdata reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)
    set(hObject, 'Enable', 'Off');
    handles = segedit_update(handles);
    set(hObject, 'Enable', 'on');
925 drawnow
    segedit('MenuItemSave_Callback', hObject, eventdata, handles);
    guidata(hObject, handles);

    % guidata(hObject, handles);
930

    function handles = segedit_update(handles)
    plottedit off
    updatestatus(handles, 'Updating mask');
935 drawnow

    im = handles.image;
    sz = handles.seg.imsz;
940 nColorCur = handles.uiprops.color.cur;
    nColorAdd = handles.uiprops.color.add;
    nColorAddGlobal = handles.uiprops.color.addglobal;
    nColorErase = handles.uiprops.color.erase;
    nColorEraseGlobal = handles.uiprops.color.eraseglobal;
945 nObjLineLnWidth = handles.uiprops.linewidth.line;
    nObjAreaLnWidth = handles.uiprops.linewidth.area;

    BCur = zeros(sz);
    BAdd = BCur;
950 BErs = BCur;

    BAddGbl = handles.maskglobal.add;
    BErsGbl = handles.maskglobal.erase;

955 % pull in the line object handles
    hObjs = get(handles.axes1, 'children');

    hLines = findobj(hObjs, 'flat', 'type', 'line');
    hLinesCur = findobj(hLines, 'flat', 'color', nColorCur);
960 hLinesAdd = findobj(hLines, 'flat', 'color', nColorAdd);

    hLinesErsArea = findobj(hLines, 'flat', ...
                           'color', nColorErase, ...
                           '-and', 'linewidth', nObjAreaLnWidth);
965 hLinesErsLine = findobj(hLines, 'flat', ...
                           'color', nColorErase, ...

```

```

                                '-and', 'linewidth', nObjLineLnWidth);

% process added regions
970 Bon      = zeros(sz);
    xy      = get([hLinesCur; hLinesAdd], {'xdata', 'ydata'});
    Bon(sub2ind(sz, round([xy{:, 2}]), round([xy{:, 1}]))) = 1;
    Bon      = imfill(Bon, 'holes');

975 % process erased regions
    Boffa    = zeros(sz);
    xy      = get(hLinesErsArea, {'xdata', 'ydata'});
    Boffa(sub2ind(sz, round([xy{:, 2}]), round([xy{:, 1}]))) = 1;
    Boffa    = imfill(Boffa, 'holes');
980
% process erased lines
    Boffl    = zeros(sz);
    xy      = get(hLinesErsLine, {'xdata', 'ydata'});

985 for i = 1:size(xy, 1),
        Boffl = Boffl | line2mask(sz, [xy{i, 1}], [xy{i, 2}]);
    end;
% Boffl(sub2ind(sz, round([xy{:, 2}]), round([xy{:, 1}]))) = 1;
    Boffl    = imdilate(Boffl, strel('square', 2));
990
    BE = Bon & ~(Boffa | Boffl);

% process any global region edits
if (~isempty(BAddGbl) ...
995     || ~isempty(BErsGbl) ...
    && (sum(BAddGbl(:)) > 0 ...
    || sum(BErsGbl(:)) > 0),
    button = questdlg([...
1000         'There are global regions to process.'...
            ' How do you wish to proceed?'],...
            'Edit All', 'Process All', ...
            'Process Current Only', ...
            'Process Current Only');
    if strcmpi(button, 'Process All'),
1005         % this could take a while. display a waitbar to update progress
            hProg = waitbar(0, 'Processing Global Edit Operation');
            nFiles = length(handles.files);
            tLoopElaps = [];

1010         for i = 1:nFiles,
                tLoopInit = clock;
                updateprogress(hProg, i, nFiles, tLoopElaps);

                waitbar(i/nFiles, hProg, 'Processing Global Edit Operation');
1015                 fpath = [handles.cwd filesep handles.files{i}];

                load(fpath, 'seg');
                handles.seg = seg;
                msk = bndidx2mask(seg.imesize, seg.objallidx);

1020                 % process global adds if there are any
                if sum(BAddGbl(:)) > 0,
                    msk = msk | BAddGbl;

```

```

        end;
1025
        % process global erases ... note regions that touch the global
        % erase mask are also removed to keep the resultant mask
        % 'clean'
        if sum(BErsGbl(:)) > 0,
1030            L = bwlabeln(msk);
            oid = unique(immultiply(L, BErsGbl));
            for j = 2:length(oid),
                msk(find(L == oid(j))) = 0;
            end;
1035        end;

        seg.objallidx = find(bwperim(msk));
        seg.objqntidx = find(bwperim(imclearborder(msk)));

1040        save(fpath, 'seg');

        tLoopElaps = [tLoopElaps etime(clock, tLoopInit)];
        end;
        close(hProg);
1045
        % refresh the currently displayed file
        handles= segedit_loadfile(handles, handles.cwd, handles.file);
        segedit_load(handles);

1050        % defer image update to after all
        % procesing is finished
        % segedit_disp(handles);

        segedit_browseinit(handles);
1055    else,

        % just work on the current image
        if sum(BAddGbl(:)) > 0,
            BE = BE | BAddGbl;
1060        end;
        if sum(BErsGbl(:)) > 0,
            L = bwlabeln(BE);
            oid = unique(immultiply(L, BErsGbl));
            for j = 2:length(oid),
1065                BE(find(L == oid(j))) = 0;
            end;
        end;

        end;
1070 end;

        handles.maskall.curr = BE;
        handles.maskqnt.curr = imclearborder(BE);

1075 delete(hLines);

        axes(handles.axes1);
        hold on
        segedit_disp_mask(handles);
1080

```

```

updatestatus(handles, '');
checkSaved(handles);
figure(handles.figure1);

1085 % -----
function segedit_deleteobj(hObj, handles)
im = handles.image;
sz = handles.seg.imsz;
1090 % process deleted region
Bd      = zeros(sz);
xy      = get(hObj, {'xdata', 'ydata'});
Bd(sub2ind(sz, round([xy{:}, 2]), round([xy{:}, 1]))) = 1;
1095 Bd      = imfill(Bd, 'holes');

% check to see what object the user is deleting
sColor = num2str(get(hObj, 'color'));
switch sColor,
1100 case num2str(handles.uiprops.color.cur),
% the object exists in the data file, the mask should be saved for
% changes to be preserved.

handles.maskall.curr = handles.maskall.curr & ~Bd;
1105 handles.maskqnt.curr = imclearborder(handles.maskall.curr);

delete(hObj);

case {num2str(handles.uiprops.color.add), ...
1110 num2str(handles.uiprops.color.erase)},
% the object is a newly added editing region that the user is
% removing. this does not affect the change state of the mask so
% do nothing special.

1115 delete(hObj);

case {num2str(handles.uiprops.color.addglobal), ...
num2str(handles.uiprops.color.eraseglobal)},
% the user is removing an object on the global mask layer. this
1120 % should not affect the saved state of the mask, but the global
% editing mask needs to be updated.

switch sColor,
case num2str(handles.uiprops.color.addglobal),
1125 handles.maskglobal.add = handles.maskglobal.add & ~Bd;
case num2str(handles.uiprops.color.eraseglobal),
handles.maskglobal.erase = handles.maskglobal.erase & ~Bd;
end;

1130 delete(hObj);

end;
checkSaved(handles);
guidata(handles.figure1, handles);
1135

% --- Executes on button press in tbDisp.

```



```

function tbDisp_Callback(hObject, eventdata, handles)
% hObject    handle to tbDisp (see GCBO)
1140 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of tbDisp
state = get(hObject, 'Value');
1145 strs = get(hObject, 'UserData');
set(hObject, 'String', strs(state+1));

if state == 0,
    % Mask on, the default
1150 set(findobj('type', 'line'), 'visible', 'on');
elseif state == 1,
    % Mask off
    set(findobj('type', 'line'), 'visible', 'off');
end;
1155
% handles key press events
function segedit_keypress(src, event, varargin)
handles = guidata(gcf);

1160 sMod = event.Modifier;
sKey = event.Key;

switch sKey,
    case {'delete', 'decimal'},
1165     if strcmpi(get(gco, 'type'), 'line'),
        % delete(gco);
        segedit_deleteobj(gco, handles);
    end;
    case {'rightarrow', 'space', 'numpad6'},
1170     pbBrowseNext_Callback(handles.pbBrowseNext, [], handles);
    case {'leftarrow', 'backspace', 'numpad4'},
        pbBrowsePrev_Callback(handles.pbBrowsePrev, [], handles);
    case 'add',
        pbAdd_Callback(handles.pbErase, [], handles);
1175 case 'subtract',
        pbErase_Callback(handles.pbErase, [], handles);
    case {'f12', 'return'},
        pbUpdateSave_Callback(handles.pbUpdateSave, [], handles);
    case {'multiply', 'm'},
1180     set(handles.tbDisp, 'value', ...
        ~logical(get(handles.tbDisp, 'value')));
        tbDisp_Callback(handles.tbDisp, [], handles);
    otherwise,
end;
1185

function segedit_onclick(hObject, varargin)
sCurSelState = get(hObject, 'selected');
set(findobj('selected', 'on'), 'selected', 'off');
1190 if strcmpi(sCurSelState, 'off'),
    set(hObject, 'selected', 'on', 'selectionhighlight', 'on');
end;

function segedit_showargs(varargin)

```

```

1195 for i=1:length(varargin),
        varargin{i}
    end;

1200 % -----
    function MenuFileUpdateSave_Callback(hObject, eventdata, handles)
        % hObject    handle to MenuFileUpdateSave (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    structure with handles and user data (see GUIDATA)
1205

        % --- Executes on selection change in pmnEraseType.
1210 function pmnEraseType_Callback(hObject, eventdata, handles)
        % hObject    handle to pmnEraseType (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    structure with handles and user data (see GUIDATA)

1215 % Hints: contents = get(hObject,'String') returns pmnEraseType contents as
        %         cell array contents{get(hObject,'Value')} returns selected item
        %         from pmnEraseType

1220 % --- Executes during object creation, after setting all properties.
    function pmnEraseType_CreateFcn(hObject, eventdata, handles)
        % hObject    handle to pmnEraseType (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    empty - handles not created until after all CreateFcns called
1225

        % Hint: popupmenu controls usually have a white background on Windows.
        %         See ISPC and COMPUTER.
        if ispc && isequal(get(hObject,'BackgroundColor'), ...
            get(0,'defaultUicontrolBackgroundColor'))
1230     set(hObject,'BackgroundColor','white');
        end

1235

        % --- Executes on selection change in pmnFilterType.
    function pmnFilterType_Callback(hObject, eventdata, handles)
        % hObject    handle to pmnFilterType (see GCBO)
        % eventdata  reserved - to be defined in a future version of MATLAB
1240 % handles    structure with handles and user data (see GUIDATA)

        % Hints: contents = get(hObject,'String') returns pmnFilterType contents as
        %         cell array contents{get(hObject,'Value')} returns selected item
        %         from pmnFilterType
1245

        % --- Executes during object creation, after setting all properties.
    function pmnFilterType_CreateFcn(hObject, eventdata, handles)
        % hObject    handle to pmnFilterType (see GCBO)
1250 % eventdata  reserved - to be defined in a future version of MATLAB
        % handles    empty - handles not created until after all CreateFcns called

```

```

% Hint: popupmenu controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
1255 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

1260

function edtFilterMin_Callback(hObject, eventdata, handles)
% hObject    handle to edtFilterMin (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1265 % handles  structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtFilterMin as text
%        str2double(get(hObject,'String')) returns contents of edtFilterMin
%        as a double
1270

% --- Executes during object creation, after setting all properties.
function edtFilterMin_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtFilterMin (see GCBO)
1275 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
1280 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end

1285

function edtFilterMax_Callback(hObject, eventdata, handles)
% hObject    handle to edtFilterMax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1290 % handles  structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edtFilterMax as text
%        str2double(get(hObject,'String')) returns contents of edtFilterMax
%        as a double
1295

% --- Executes during object creation, after setting all properties.
function edtFilterMax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtFilterMax (see GCBO)
1300 % eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%     See ISPC and COMPUTER.
1305 if ispc && isequal(get(hObject,'BackgroundColor'), ...
        get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
end

```

```

1310
% --- Executes on button press in pbFilterHistogram.
function pbFilterHistogram_Callback(hObject, eventdata, handles)
% hObject    handle to pbFilterHistogram (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1315 % handles  structure with handles and user data (see GUIDATA)

% display a histogram for the filter value specified in pmnFilterType
sFilterType = get(handles.pmnFilterType, 'String');
sFilterType = sFilterType{get(handles.pmnFilterType, 'Value')};
1320
data = segedit_getfilterdata(handles, handles.maskall.curr, sFilterType);
figure,
[n, x] = hist(data, sqrt(length(data)));
bar(x, n);
1325 xlabel(sFilterType);
ylabel('Counts');

% -----
function [data, L] = segedit_getfilterdata(handles, B, sFilterType)
1330 L = bwlabeln(B);
I = handles.image;

iNumObjs = length(unique(L)) - 1;

1335 switch lower(sFilterType),
    case {'area', 'perimeter'},
        S = regionprops(L, sFilterType);
        data = [S(:).(sFilterType)];
    case {'intensity', 'variance'},
1340         data = [];
        for i = 1:iNumObjs,
            switch lower(sFilterType),
                case 'intensity',
                    data = [data, mean2(I(find(L == i)))]];
1345                 case 'variance',
                    data = [data, std2(I(find(L == i)))]];
            end;
        end;
    otherwise,
1350 end;

% --- Executes on button press in pbFilter.
function pbFilter_Callback(hObject, eventdata, handles)
1355 % hObject    handle to pbFilter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

bApplyAll = get(handles.chkFilterApplyAll, 'value');
1360
if bApplyAll,
    % Display a warning notifying the user that this operation updates all
    % files in the current working directory without an undo
    button = questdlg(['...
1365         'This will irreversibly modify all files in the current '...

```

```

        'working directory. Do you wish to continue?'],...
        'Morph All', 'Yes', 'No', 'No');
if strcmpi(button, 'yes'),
    % this could take a while.  display a waitbar to update progress
1370    hProg = waitbar(0, 'Processing Global Filter Operation');
        nFiles = length(handles.files);
        tLoopElaps = [];

        for i = 1:nFiles,
1375            tLoopInit = clock;
                updateprogress(hProg, i, nFiles, tLoopElaps);

                waitbar(i/nFiles, hProg, 'Processing Global Filter Operation');
                fpath = [handles.cwd filesep handles.files{i}];
1380
                load(fpath, 'seg');
                handles.seg = seg;
                handles.image = segedit_getimg(handles);

1385                msk = bndidx2mask(seg.imsz, seg.objallidx);
                msk = segedit_filterobjs(handles, msk);

                seg.objallidx = find(bwperim(msk));
                seg.objqntidx = find(bwperim(imclearborder(msk)));
1390
                save(fpath, 'seg');

                tLoopElaps = [tLoopElaps etime(clock, tLoopInit)];
            end;
1395            close(hProg);

            % refresh the currently displayed file
            handles= segedit_loadfile(handles, handles.cwd, handles.file);
            segedit_load(handles);
1400            segedit_disp(handles);
            segedit_browseinit(handles);
        end;

    else,
1405        handles.maskall.curr = segedit_filterobjs(handles, ...
                handles.maskall.curr);
        handles.maskqnt.curr = imclearborder(handles.maskall.curr);

        hObjs = get(handles.axes1, 'children');
1410        hLines = findobj(hObjs, 'flat', 'type', 'line');
        delete(hLines);

        axes(handles.axes1);
        hold on
1415        segedit_disp_mask(handles);

        checkSaved(handles);
        guidata(hObject, handles);
    end;
1420

% -----

```

```

function B = segedit_filterobjs(handles, msk)
% get the filter type
1425 sFilterType = get(handles.pmnFilterType, 'String');
    sFilterType = sFilterType{get(handles.pmnFilterType, 'Value')};

% get the filter range
    nMax = str2double(get(handles.edtFilterMax, 'String'));
1430 nMin = str2double(get(handles.edtFilterMin, 'String'));

% get the filter method
    bInclude = get(handles.radFilterInclude, 'Value');
    bExclude = get(handles.radFilterExclude, 'Value');
1435
    [data, L] = segedit_getfilterdata(handles, msk, sFilterType);

    B = zeros(size(L));
    vObjs = find(data >= nMin & data <= nMax);
1440 for i = 1:length(vObjs),
        B(find(L == vObjs(i))) = 1;
    end;

    if bInclude,
1445     % keep the objects that match the filter range
        B = msk & B;

    elseif bExclude,
        % remove the objects that match the filter range
1450     B = msk & ~B;

    end;

1455 % --- Executes on button press in radFilterInclude.
function radFilterInclude_Callback(hObject, eventdata, handles)
% hObject    handle to radFilterInclude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1460
% Hint: get(hObject,'Value') returns toggle state of radFilterInclude

% --- Executes on button press in radFilterExclude.
1465 function radFilterExclude_Callback(hObject, eventdata, handles)
% hObject    handle to radFilterExclude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

1470 % Hint: get(hObject,'Value') returns toggle state of radFilterExclude

1475 % --- Executes on button press in pbFilterPreview.
function pbFilterPreview_Callback(hObject, eventdata, handles)
% hObject    handle to pbFilterPreview (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

1480     B = segedit_filterobjs(handles, handles.maskall.curr);

        hObjs = get(handles.axes1, 'children');
        hLines = findobj(hObjs, 'flat', 'type', 'line');
1485 hLines = findobj(hLines, 'flat', 'color', handles.uiprops.color.preview);
        delete(hLines);

        axes(handles.axes1);
        hold on
1490 BND = bwboundaries(B, 'noholes');
        for i = 1:length(BND),
            plot(BND{i}(:, 2), BND{i}(:,1), ...
                'color', handles.uiprops.color.preview, ...
                'linestyle', '-', ...
1495             'linewidth', handles.uiprops.linewidth.area)
            hold on
        end;

1500 % --- Executes on button press in pbFilterPreviewClear.
        function pbFilterPreviewClear_Callback(hObject, eventdata, handles)
            % hObject    handle to pbFilterPreviewClear (see GCBO)
            % eventdata  reserved - to be defined in a future version of MATLAB
            % handles    structure with handles and user data (see GUIDATA)
1505 segedit_clearpreview(handles);

        % -----
        function segedit_clearpreview(handles)
1510 hObjs = get(handles.axes1, 'children');
            hLines = findobj(hObjs, 'flat', 'type', 'line');
            hLines = findobj(hLines, 'flat', 'color', handles.uiprops.color.preview);
            delete(hLines);

1515

        % --- Executes on button press in pbMorphPreviewClear.
        function pbMorphPreviewClear_Callback(hObject, eventdata, handles)
            % hObject    handle to pbMorphPreviewClear (see GCBO)
            % eventdata  reserved - to be defined in a future version of MATLAB
1520 % handles    structure with handles and user data (see GUIDATA)
            segedit_clearpreview(handles);

1525 % --- Executes on button press in pbMorphPreview.
        function pbMorphPreview_Callback(hObject, eventdata, handles)
            % hObject    handle to pbMorphPreview (see GCBO)
            % eventdata  reserved - to be defined in a future version of MATLAB
            % handles    structure with handles and user data (see GUIDATA)
1530
            B = segedit_morphobjs(handles, handles.maskall.curr);

            hObjs = get(handles.axes1, 'children');
            hLines = findobj(hObjs, 'flat', 'type', 'line');
1535 hLines = findobj(hLines, 'flat', 'color', handles.uiprops.color.preview);
            delete(hLines);

```

```

axes(handles.axes1);
hold on
1540 BND = bwboundaries(B, 'noholes');
    for i = 1:length(BND),
        plot(BND{i}(:, 2), BND{i}(:,1), ...
            'color', handles.uiprops.color.preview, ...
            'linestyle', '-', ...
1545         'linewidth', handles.uiprops.linewidth.area)
        hold on
    end;

% --- Executes on button press in chkFilterApplyAll.
1550 function chkFilterApplyAll_Callback(hObject, eventdata, handles)
% hObject    handle to chkFilterApplyAll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

1555 % Hint: get(hObject,'Value') returns toggle state of chkFilterApplyAll

1560 % --- Executes on button press in chkMorphApplyAll.
function chkMorphApplyAll_Callback(hObject, eventdata, handles)
% hObject    handle to chkMorphApplyAll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
1565
% Hint: get(hObject,'Value') returns toggle state of chkMorphApplyAll

1570
% --- Executes when figure1 is resized.
function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
1575 % handles    structure with handles and user data (see GUIDATA)

if isfield(handles, 'figure1'),
% objects that stretch:
% axes1, the main display axis, vertical, horizontal
1580 % sldBrowse, the browsing slider, horizontal
% edtStatus, the status message field, horizontal
% panel8, the blank spacer panel, vertical

% all other objects dimensions should remain fixed
1585 % aside from stretched positions, all other coordinates should remain fixed
% relative to the figure window edge

% position all objects lower left corners to where they should be relative
% to the figure lower left
1590
% get the current figure position
cfpos = get(handles.figure1, 'position');

```



```

% objects to move to upper left corner
1595 % calculate distance offsets to move lower right corners by measuring how
% much the figure has grown
offset = cfpos - handles.uisize;
offset = [offset(3:4), 0, 0];

1600 h = [ handles.tbDisp;
         handles.uipanel9;
         handles.uipanel2;
         handles.uipanel7;
         ];
1605 for i = 1:length(h),
        set(h(i), 'position', get(h(i), 'position') + offset);
    end;

% stretch objects that need h/v stretch
1610 stretch = circshift(offset, [0 2]);
h = [ handles.axes1;
      ];
for i = 1:length(h),
    set(h(i), 'position', get(h(i), 'position') + stretch);
1615 end;

% stretch objects that need v stretch
stretchv = stretch;
stretchv(3) = 0;
1620 h = [ handles.uipanel8;
          ];
for i = 1:length(h),
    set(h(i), 'position', get(h(i), 'position') + stretchv);
end;
1625

% stretch objects that need h stretch
stretchh = stretch;
stretchh(4) = 0;
1630 h = [ handles.sldBrowse;
          handles.edtStatus;
          ];
for i = 1:length(h),
    set(h(i), 'position', get(h(i), 'position') + stretchh);
end;
1635

% objects to move left only
offset(2) = 0;
h = [ handles.uipanel8;
      handles.uipanel3;
1640     handles.pbBrowsePrev;
      handles.edtBrowse;
      handles.pbBrowseNext;
      ];
for i = 1:length(h),
1645     set(h(i), 'position', get(h(i), 'position') + offset);
end;

handles.uisize = cfpos;
guidata(hObject, handles);
1650 end; % if figure is drawn

```

```
% --- Executes on button press in chkMaskApplyAll.  
function chkMaskApplyAll_Callback(hObject, eventdata, handles)  
1655 % hObject    handle to chkMaskApplyAll (see GCBO)  
% eventdata reserved - to be defined in a future version of MATLAB  
% handles      structure with handles and user data (see GUIDATA)  
  
% Hint: get(hObject,'Value') returns toggle state of chkMaskApplyAll  
1660
```

B.8 Complete source code for TRACKEDIT

```

function varargout = trackedit(varargin)
% TRACKEDIT M-file for trackedit.fig
%   TRACKEDIT, by itself, creates a new TRACKEDIT or raises the existing
%   singleton*.
5 %
%   H = TRACKEDIT returns the handle to a new TRACKEDIT or the handle to
%   the existing singleton*.
%
%   TRACKEDIT('CALLBACK',hObject,eventData,handles,...) calls the local
10 %   function named CALLBACK in TRACKEDIT.M with the given input
%   arguments.
%
%   TRACKEDIT('Property','Value',...) creates a new TRACKEDIT or raises
%   the existing singleton*. Starting from the left, property value
15 %   pairs are applied to the GUI before trackedit_OpeningFunction gets
%   called. An unrecognized property name or invalid value makes
%   property application stop. All inputs are passed to
%   trackedit_OpeningFcn via varargin.
%
20 %   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

25 % Edit the above text to modify the response to help trackedit

% Last Modified by GUIDE v2.5 11-Jul-2006 00:32:09

% Begin initialization code - DO NOT EDIT
30 gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @trackedit_OpeningFcn, ...
                  'gui_OutputFcn',  @trackedit_OutputFcn, ...
35                  'gui_LayoutFcn', [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

40 if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
45 end
% End initialization code - DO NOT EDIT

% --- Executes just before trackedit is made visible.
50 function trackedit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

55 % varargin    command line arguments to trackedit (see VARARGIN)

    % Choose default command line output for trackedit
    handles.output = hObject;

60 hUI = findobj(handles.figure1, ...
                '-depth', 1, ...
                'type', 'uicontrol', ...
                '-or', 'type', 'uipanel', ...
                '-or', 'type', 'axes');
65 hUI = [handles.figure1; hUI];
    set(hUI, 'units', 'pixels');
    handles.uisize = get(handles.figure1, 'position');

    % Application defaults
70 opts = getopt(varargin);

    handles.uidata.infofile = parseopts('file', opts, '');
    handles.uidata.cwd = cd;
    handles.uidata.overlay = '';
75 handles.uidata.image = [];

    % default colors
    handles.uidata.colors = struct(...
        'segmask', [1 0 0], ...
80     'objids', [0.5 1 0.5], ...
        'missingmask', [1 1 0], ...
        'missingobjids', [1 0.5 0] ...
    );

85 % display everything
    handles.uidata.display = struct(...
        'image', 1, ...
        'seg', 1, ...
        'objid', 1, ...
90     'missseg', 1, ...
        'missobjid', 1 ...
    );
    set([ ...
        handles.chkDispImage;
95     handles.chkDispSegMask;
        handles.chkDispObjIds;
        handles.chkDispMissingObjMask;
        handles.chkDispMissingObjIds;
        ] ...
100    , 'value', 1);

    handles.rundata = [];

    set(handles.figure1, 'keypressfcn', @trackedit_keypress);
105
    % Update handles structure
    guidata(hObject, handles);

    if ~isempty(handles.uidata.infofile),
110     [fpath, fname, fext, fver] = fileparts(handles.uidata.infofile);

```

```

        if isempty(fpath), fpath = cd; end;
        fname = [fname fext];
115     MenuFileOpen_Callback(hObject, eventdata, handles, ...
                                'fpath', fpath, 'fname', fname);
    end;
    set(handles.axes1, 'visible', 'off');

120

    % UIWAIT makes trackedit wait for user response (see UIRESUME)
    % uiwait(handles.figure1);

125
    % --- Outputs from this function are returned to the command line.
    function varargout = trackedit_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % hObject    handle to figure
130 % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Get default command line output from handles structure
    varargout{1} = handles.output;
135

    % --- Executes on slider movement.
    function sldBrowse_Callback(hObject, eventdata, handles)
    % hObject    handle to sldBrowse (see GCBO)
140 % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Hints: get(hObject,'Value') returns position of slider
    %          get(hObject,'Min') and get(hObject,'Max') to determine range of slider
145
    handles.uidata.imgid = floor(get(hObject, 'value'));
    set(handles.txtBrowse, ...
        'string', sprintf('%d of %d', ...
                            handles.uidata.imgid, ...
150                            length(handles.rundata.info.indices)));
    set(handles.edtBrowse, 'string', num2str(handles.uidata.imgid));
    trackedit_disp(handles);

    guidata(hObject, handles);
155

    % --- Executes during object creation, after setting all properties.
    function sldBrowse_CreateFcn(hObject, eventdata, handles)
160 % hObject    handle to sldBrowse (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    empty - handles not created until after all CreateFcns called

    % Hint: slider controls usually have a light gray background.
165 if isequal(get(hObject,'BackgroundColor'), ...
                get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
    end

```

```

170
% --- Executes on button press in pbBrowsePrev.
function pbBrowsePrev_Callback(hObject, eventdata, handles)
% hObject    handle to pbBrowsePrev (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
175 % handles   structure with handles and user data (see GUIDATA)

    currid = get(handles.sldBrowse, 'value');
    if currid > get(handles.sldBrowse, 'min'),
        set(handles.sldBrowse, 'value', currid - 1);
180     sldBrowse_Callback(handles.sldBrowse, eventdata, handles);
    end;

% --- Executes on button press in pbBrowseNext.
function pbBrowseNext_Callback(hObject, eventdata, handles)
185 % hObject    handle to pbBrowseNext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

    currid = get(handles.sldBrowse, 'value');
190 if currid < get(handles.sldBrowse, 'max'),
        set(handles.sldBrowse, 'value', currid + 1);
        sldBrowse_Callback(handles.sldBrowse, eventdata, handles);
    end;

195
function edtBrowse_Callback(hObject, eventdata, handles)
% hObject    handle to edtBrowse (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
200
% Hints: get(hObject,'String') returns contents of edtBrowse as text
%        str2double(get(hObject,'String')) returns contents of edtBrowse as
%        a double
handles.uidata.imgid = floor(str2num(get(hObject, 'string')));
205 set(handles.txtBrowse, ...
        'string', sprintf('%d of %d', ...
                           handles.uidata.imgid, ...
                           length(handles.rundata.info.indices)));
    set(handles.sldBrowse, 'value', handles.uidata.imgid);
210 trackedit_disp(handles);

% --- Executes during object creation, after setting all properties.
function edtBrowse_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edtBrowse (see GCBO)
215 % eventdata  reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
220 if ispc && isequal(get(hObject,'BackgroundColor'), ...
                    get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
225

```

```

% -----
function MenuFileOpen_Callback(hObject, eventdata, handles, varargin)
% hObject    handle to MenuFileOpen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
230 % handles   structure with handles and user data (see GUIDATA)
    opts = getopts(varargin);
    fname = parseopts('fname', opts, '');
    fpath = parseopts('fpath', opts, '');

235 if isempty(fname) & isempty(fpath),
        [fname fpath] = uigetfile('*.mat', 'Select Run-time Info File', ...
                                'imsqt');
    end;
    if fname ~= 0,
240     handles.uidata.infofile = [fpath filesep fname];
        handles.uidata.cwd = fpath;
        cd(fpath);

        load(handles.uidata.infofile, 'data');
245     handles.rundata = data;

        % determine the segmentation channel to overlay
        cvChannels = handles.rundata.channels;
250     [i j] = ind2sub(size(cvChannels), find(strcmpi(cvChannels, 'seg')));
        handles.uidata.overlay = cvChannels{i, 1};

        % determine how many images there are to browse through
        vIdx = handles.rundata.info.indices;
255     nTotImgs = length(vIdx);
        handles.uidata.imgid = vIdx(1);
        set(handles.sldBrowse, 'min', vIdx(1));
        set(handles.sldBrowse, 'max', vIdx(end));
        set(handles.sldBrowse, 'sliderstep', [1/nTotImgs 10/nTotImgs]);
260     set(handles.sldBrowse, 'value', vIdx(1));
        set(handles.txtBrowse, 'string', sprintf('%d of %d', 1, nTotImgs));
        set(handles.edtBrowse, 'string', num2str(vIdx(1)));

        % get the current image and object ids and display
265     trackedit_disp(handles);
        %set(handles.axes1, 'visible', 'on');

        guidata(hObject, handles);
    end;
270 % -----
function MenuFileSave_Callback(hObject, eventdata, handles)
% hObject    handle to MenuFileSave (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
275 % handles   structure with handles and user data (see GUIDATA)

% -----
function MenuFileSaveAs_Callback(hObject, eventdata, handles)
280 % hObject    handle to MenuFileSaveAs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```

```

285 % -----
function MenuFileQuit_Callback(hObject, eventdata, handles)
% hObject    handle to MenuFileQuit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
290 guidata(hObject, handles);

delete(handles.figure1)

% -----
295 function MenuFile_Callback(hObject, eventdata, handles)
% hObject    handle to MenuFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

300 % -----
function I = trackedit_getimg(handles, varargin)
iROI = hashval(handles.rundata.channels, handles.uidata.overlay);
iROI = hashval(handles.rundata.rois, iROI{6});
iROI = iROI{1};
305
I = getimg( ...
        handles.uidata.overlay, ...
        handles.rundata.info.locid, ...
        handles.uidata.imgid, ...
310        'basepath', trackedit_getexptpath(handles));
if ~isempty(iROI),
    I = imcrop(I, iROI);
end;

315 % -----
function sExptPath = trackedit_getexptpath(handles)
sExptPath = [];
csStrings = {handles.rundata.info.path, ...
             handles.rundata.info.prefix, handles.rundata.info.id};
320
% only assign the path if all the fields have been set
if isempty(strmatch('', csStrings, 'exact')),
    sExptPath = sprintf('%s%s%s', csStrings{1}, filesep, csStrings{2:3});
end;
325 sExptPath = handles.rundata.info.path;

% -----
function trackedit_disp(handles)
axes(handles.axes1);
330 hold off
trackedit_disp_image(handles);
hold on
trackedit_disp_mask(handles);
trackedit_disp_objids(handles);
335 set(handles.axes1, 'xtick', [], 'ytick', [], 'box', 'on', 'color', [0 0 0]);

% -----
function trackedit_disp_image(handles)
hold off

```



```

340 axes(handles.axes1);
    img = trackedit_getimg(handles);
    imshow(img, [], 'InitialMagnification', 'fit');
    chkDispImage_Callback(handles.chkDispImage, [], handles);
    axis on
345 set(handles.axes1, 'xtick', [], 'ytick', [], 'box', 'on', 'color', [0 0 0]);

    % -----
    function trackedit_disp_mask(handles)
    axes(handles.axes1);
350 sFileName = [ trackedit_getexptpath(handles), ...
                filesep, handles.rundata.segment.path, ...
                filesep, sprintf('%03d', handles.rundata.info.locid), ...
                filesep, sprintf('%04d.mat', handles.uidata.imgid) ...
                ];
355     sFileNamePrev = [ trackedit_getexptpath(handles), ...
                       filesep, handles.rundata.segment.path, ...
                       filesep, sprintf('%03d', handles.rundata.info.locid), ...
                       filesep, sprintf('%04d.mat', handles.uidata.imgid-1) ...
                       ];
360     ];

    seg = load(sFileName, 'seg');
    seg = seg.seg;

365 clr.segmask = handles.uidata.colors.segmask;
    clr.missingmask = handles.uidata.colors.missingmask;

    msk = bndidx2mask(seg.imsz, seg.objqntidx);
    BND = bwboundaries(msk, 'noholes');
370 for i = 1:length(BND),
        plot(BND{i}(:, 2), BND{i}(:,1), 'color', clr.segmask, ...
              'linestyle', '-', 'linewidth', 1)

        hold on
    end;
375 chkDispSegMask_Callback(handles.chkDispSegMask, [], handles);

    % marks changes in object locations
    if handles.uidata.imgid > 0 && exist(sFileNamePrev, 'file'),
        seg = load(sFileNamePrev, 'seg');
380     seg = seg.seg;
        mskprev = bndidx2mask(seg.imsz, seg.objqntidx);

        mskdiff = imopen(xor(msk, mskprev), strel('disk', 5));
        BND = bwboundaries(mskdiff, 'noholes');
385     for i = 1:length(BND),
            plot(BND{i}(:, 2), BND{i}(:,1), 'color', clr.missingmask, ...
                  'linestyle', '-', 'linewidth', 2)

            hold on
        end;
390     chkDispMissingObjMask_Callback(handles.chkDispMissingObjMask, [], ...
                                      handles);

    end;

395 hold off
    axis([0 seg.imsz(2) 0 seg.imsz(1)])

```

```

% -----
400 function trackedit_disp_objids(handles)
% get the object id and location data from the tracking data by image
sFileName = [ trackedit_getexptpath(handles), ...
             filesep, handles.rundata.track.path, ...
             filesep, sprintf('%03d', handles.rundata.info.locid), ...
405             filesep, 'byimg', ...
             filesep, sprintf('%04d.img.dat', handles.uidata.imgid) ...
            ];

sFileNamePrev = [ trackedit_getexptpath(handles), ...
410                filesep, handles.rundata.track.path, ...
                filesep, sprintf('%03d', handles.rundata.info.locid), ...
                filesep, 'byimg', ...
                filesep, sprintf('%04d.img.dat', handles.uidata.imgid-1)...
            ];

415 objects = importdata(sFileName);
data = objects.data;
cols = objects.colheaders;

handles.uidata.objects = objects.data;
420 handles.uidata.objectcols = objects.colheaders;

xpos = getdatacol('centx', data, cols);
ypos = getdatacol('centy', data, cols);
objid = getdatacol('newobjid', data, cols);
425
tmp = cell(size(objid));
for i = 1:length(objid),
    tmp{i} = num2str(objid(i));
end;
430 objidstr = tmp;

clr.objids = handles.uidata.colors.objids;
clr.missingobjids = handles.uidata.colors.missingobjids;

435 axes(handles.axes1);
handles.uidata.objecthandles = text(xpos, ypos, objidstr, ...
    'color', clr.objids, ...
    'fontname', 'monotype', ...
    'fontsize', 8, ...
440    'verticalalign', 'middle', ...
    'horizontalalign', 'center' ...
);
chkDispObjIds_Callback(handles.chkDispObjIds, [], handles);

445 % marks changes in object locations
if handles.uidata.imgid > 0 && exist(sFileNamePrev, 'file'),
    objectsprev = importdata(sFileNamePrev);
    objidprev = getdatacol('newobjid', objectsprev);

450    objsdiff = ~ismember(objidprev, objid);

xpos = getdatacol('centx', objectsprev);
ypos = getdatacol('centy', objectsprev);

```

```

455   objid = objidprev(objsdiff);
      xpos = xpos(objsdiff);
      ypos = ypos(objsdiff);

      tmp = cell(size(objid));
460   for i = 1:length(objid),
         tmp{i} = num2str(objid(i));
      end;
      objidstr = tmp;

465   axes(handles.axes1);
      handles.uidata.objecthandles = text(xpos, ypos, objidstr, ...
          'color', clr.missingobjids, ...
          'fontname', 'monotype', ...
          'fontsize', 8, ...
470   'verticalalign', 'middle', ...
          'horizontalalign', 'center' ...
      );
      chkDispMissingObjIds_Callback(handles.chkDispMissingObjIds, [], ...
                                     handles);

475   end;

% --- Executes when figure1 is resized.
480 function figure1_ResizeFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

485 if isfield(handles, 'figure1'),
% objects that stretch:
% axes1, the main display axis, vertical, horizontal
% sldBrowse, the browsing slider, horizontal
% edtStatus, the status message field, horizontal
490 % panel8, the blank spacer panel, vertical

% all other objects dimensions should remain fixed
% aside from stretched positions, all other coordinates should remain fixed
% relative to the figure window edge
495 % position all objects lower left corners to where they should be relative
% to the figure lower left

% get the current figure position
500 cfpos = get(handles.figure1, 'position');

% objects to move to upper left corner
% calculate distance offsets to move lower right corners by measuring how
% much the figure has grown
505 offset = cfpos - handles.uisize;
      offset = [offset(3:4), 0, 0];

      h = [handles.pnlDisplay];
      for i = 1:length(h),
510         set(h(i), 'position', get(h(i), 'position') + offset);

```

```

end;

% stretch objects that need h/v stretch
stretch = circshift(offset, [0 2]);
515 h = [ handles.axes1;
        ];
    for i = 1:length(h),
        set(h(i), 'position', get(h(i), 'position') + stretch);
    end;
520
% stretch objects that need v stretch
stretchv = stretch;
stretchv(3) = 0;
h = [ handles.pnlEdit;
525     ];
    for i = 1:length(h),
        set(h(i), 'position', get(h(i), 'position') + stretchv);
    end;

530 % stretch objects that need h stretch
stretchh = stretch;
stretchh(4) = 0;
h = [ handles.sldBrowse;
        ];
535 for i = 1:length(h),
        set(h(i), 'position', get(h(i), 'position') + stretchh);
    end;

% objects to move left only
540 offset(2) = 0;
h = [ handles.pnlUpdate;
        handles.txtBrowse;
        handles.pbBrowsePrev;
        handles.edtBrowse;
545     handles.pbBrowseNext;
        handles.pnlEdit;
        ];
    for i = 1:length(h),
        set(h(i), 'position', get(h(i), 'position') + offset);
550 end;

handles.uisize = cfpos;
guidata(hObject, handles);
end;
555
% handles key press events
function trackedit_keypress(src, event, varargin)
handles = guidata(gcf);

560 sMod = event.Modifier;
sKey = event.Key;

switch sKey,
    case {'delete', 'decimal'},
565         % nothing yet
    case {'rightarrow', 'space', 'numpad6'},
        pbBrowseNext_Callback(handles.pbBrowseNext, [], handles);

```

```

        case {'leftarrow', 'backspace', 'numpad4'},
            pbBrowsePrev_Callback(handles.pbBrowsePrev, [], handles);
570     case 'add',
            % nothing yet
        case 'subtract',
            % nothing yet
        case {'f12', 'return'},
575         % nothing yet
        case {'multiply', 'm'},
            % nothing yet
        otherwise,
    end;
580

% --- Executes on button press in chkDispImage.
function chkDispImage_Callback(hObject, eventdata, handles)
% hObject    handle to chkDispImage (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
585 % handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of chkDispImage
himg = findobj(get(handles.axes1, 'children'), 'type', 'image');
590
val = get(hObject, 'value');
handles.uidata.display.image = val;
if val,
    set(himg, 'visible', 'on');
595 else
    set(himg, 'visible', 'off');
end;

guidata(hObject, handles);
600

% --- Executes on button press in chkDispSegMask.
function chkDispSegMask_Callback(hObject, eventdata, handles)
% hObject    handle to chkDispSegMask (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
605 % handles   structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of chkDispSegMask
hmsk = findobj(get(handles.axes1, 'children'), ...
    'type', 'line', ...
610     '-and', 'color', handles.uidata.colors.segmask);

val = get(hObject, 'value');
handles.uidata.display.seg = val;
if val,
615     set(hmsk, 'visible', 'on');
    else
        set(hmsk, 'visible', 'off');
    end;

620 guidata(hObject, handles);

% --- Executes on button press in chkDispMissingObjMask.
function chkDispMissingObjMask_Callback(hObject, eventdata, handles)
% hObject    handle to chkDispMissingObjMask (see GCBO)

```

```

625 % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)

    % Hint: get(hObject,'Value') returns toggle state of chkDispMissingObjMask
    hmsk = findobj(get(handles.axes1, 'children'), ...
630         'type', 'line', ...
            '-and', 'color', handles.uidata.colors.missingmask);

    val = get(hObject, 'value');
    handles.uidata.display.misseg = val;
635 if val,
        set(hmsk, 'visible', 'on');
    else
        set(hmsk, 'visible', 'off');
    end;
640 guidata(hObject, handles);

    % --- Executes on button press in chkDispObjIds.
    function chkDispObjIds_Callback(hObject, eventdata, handles)
    % hObject handle to chkDispObjIds (see GCBO)
    % eventdata reserved - to be defined in a future version of MATLAB
645 % handles structure with handles and user data (see GUIDATA)

    % Hint: get(hObject,'Value') returns toggle state of chkDispObjIds
    hobj = findobj(get(handles.axes1, 'children'), ...
650         'type', 'text', ...
            '-and', 'color', handles.uidata.colors.objids);

    val = get(hObject, 'value');
    handles.uidata.display.objid = val;
655 if val,
        set(hobj, 'visible', 'on');
    else
        set(hobj, 'visible', 'off');
    end;
660 guidata(hObject, handles);

    % --- Executes on button press in chkDispMissingObjIds.
    function chkDispMissingObjIds_Callback(hObject, eventdata, handles)
    % hObject handle to chkDispMissingObjIds (see GCBO)
665 % eventdata reserved - to be defined in a future version of MATLAB
    % handles structure with handles and user data (see GUIDATA)

    % Hint: get(hObject,'Value') returns toggle state of chkDispMissingObjIds
    hobj = findobj(get(handles.axes1, 'children'), ...
670         'type', 'text', ...
            '-and', 'color', handles.uidata.colors.missingobjids);

    val = get(hObject, 'value');
    handles.uidata.display.missobjid = val;
675 if val,
        set(hobj, 'visible', 'on');
    else
        set(hobj, 'visible', 'off');
    end;
680 guidata(hObject, handles);

```

```
% --- Executes during object creation, after setting all properties.  
function chkDispImage_CreateFcn(hObject, eventdata, handles)  
685 % hObject    handle to chkDispImage (see GCBO)  
% eventdata  reserved - to be defined in a future version of MATLAB  
% handles    empty - handles not created until after all CreateFcns called
```

B.9 Complete source code for TRACKVIEW

```

function [hFig plotdata] = trackview(locid, objid, varargin)
opts = getopt('varargin');
inpperiod = parseopts('period', opts, 60);
smival = parseopts('smival', opts, 60);
5 imival = parseopts('imival', opts, 5);

bLink = parseopts('link', opts, false);
trjpath = parseopts('trjpath', opts, '/trjdata');
% qntpath = parseopts('qntpath', opts, '/objdata');
10 plots = lower(parseopts('plots', opts, {'fl2', 'area', 'var2'}));
plotset= {'fl[0-9]*', 'std[0-9]*', 'bg[0-9]*', 'var[0-9]*', ...
          'area', 'perim', 'xy'};

plotsm = parseopts('smoothing', opts, cell(size(plots)));
15 clr = parseopts('color', opts, 'b');

bGraphic = parseopts('graphic', opts, true);

20 % initialize
% glcmax = 2.0;

if bGraphic,
    hFig = figure;
25    set(hFig, 'name', sprintf('Location: %03d, Object: %04d', ...
                               locid, objid));
else
    hFig = [];
end;
30 if bLink,
    data = tracklink(locid, objid, 'trjpath', trjpath);
else
    data = importdata(sprintf('.%s%c%03d%cbyobj%c%04d.obj.dat', ...
35    trjpath, filesep, locid, filesep, filesep, objid));
end;

data.smival = smival;
data.period = inpperiod;
40 data.imival = imival;

t = getdatacol('imgid', data) * data.imival;

% hView = figure;
45 plotdata = cell(size(plots, 2), 2);
for i = 1:length(plots),
    if bGraphic,
        figure(hFig),
        subplot(2,2,i);
50    end;

    for j = 1:length(plotset),
        if regexpi(plots{i}, plotset{j}),
            break;

```



```

55     end;
    end;

    switch splotset{j},
        case {'fl[0-9]*', 'std[0-9]*', 'bg[0-9]*', 'var[0-9]*'},
60
            switch splotset{j},
                case {'fl[0-9]*', 'bg[0-9]*'},
                    [fn ch] = strread(splots{i}, '%2s%d');
                case {'std[0-9]*', 'var[0-9]*'},
65                    [fn ch] = strread(splots{i}, '%3s%d');
            end;

            % special calculation if variance is requested
            if ~strcmpi(fn, 'var'),
70                col = sprintf('%s%d', fn{1}, ch);
                ydata = getdatacol(col, data);
                ylabel = sprintf('%s-%d (AU)', upper(fn{1}), ch);
            else
                col = sprintf('std%d', ch);
75                ydata = getdatacol(col, data);
                col = sprintf('fl%d', ch);
                ydata = ydata ./ getdatacol(col, data);
                ylabel = sprintf('C_V-%d (AU)', ch);
            end;

80            xdata = t;
            xlabel = 'Time';

        case {'area', 'perim'},
85
            switch splotset{j},
                case 'area',
                    ydata = getdatacol('area', data);
                    ylabel = 'Area (Pixels)';
                case 'perim',
90                    ydata = getdatacol('perimnative', data);
                    ylabel = 'Perim. (Pixels)';
            end;

95            xdata = t;
            xlabel = 'Time';

        case 'xy',
            xdata = getdatacol('centx', data);
            ydata = getdatacol('centy', data);
100            xlabel = 'x';
            ylabel = 'y';
    end;

105    if ~isempty(splotsm{i}) && ischar(splotsm{i})
        if ismember(lower(splotsm{i}), {'dt', 'sm', 'lg'})
            win = data.smival / data.imival;
            [dt lg sm] = detrend(ydata, win*2, win/2, 'rlowess');
110
            switch lower(splotsm{i}),
                case 'dt',

```

```

        ydata = dt;
        case 'sm',
            ydata = sm;
115     case 'lg',
            ydata = lg;
        end;
    end;
end;

120
if bGraphic,
    plot(xdata, ydata, 'marker', 'none', ...
        'linestyle', '-', 'color', clr)

    xlabel(xlbl)
125     ylabel(ylbl)
        drawnow
    end;

    plotdata{i, 1} = xdata;
130     plotdata{i, 2} = ydata;
        plotdata{i, 3} = xlabel;
        plotdata{i, 4} = ylabel;
    end;

135 % figure(hFig)
    %
    % f = getdatacol('fl2', data);
    % s = getdatacol('std2', data);
    % v = s./f;
140 % b = getdatacol('bg2', data);
    % a = getdatacol('area', data);
    % x = getdatacol('centx', data);
    % y = getdatacol('centy', data);
    %
145 %
    % win = data.smival / data.imival;
    %
    % [fdt flg fsm] = detrend(f, win*2, win/2, 'rlowess');
    % [vdt vlg vsm] = detrend(v, win*2, win/2, 'rlowess');
150 % [adt alg asm] = detrend(a, win*2, win/2, 'rlowess');
    %
    %
    % subplot(221), plot(t, fsm, 'marker','none', 'linestyle','-', 'color',clr);
    % subplot(222), plot(t, alg, 'marker','none', 'linestyle','-', 'color',clr);
155 % subplot(223), plot(t, vsm, 'marker','none', 'linestyle','-', 'color',clr);
    %
    %
    % r = importdata(sprintf('.%s%c%03d.blob.dat', qntpath, filesep, locid));
    % t = getdatacol('imgid', r) * data.imival;
160 % r = normrange(getdatacol('bglvll', r))*glcmax;

    % subplot(224), plot(t, r, 'marker','none', 'linestyle','-', 'color',clr);
    % ax = axis;

165 %
    subplot(224), plot(x, y, 'marker','.', 'linestyle','-', 'color',clr);
    %
    hold on
    plot(x(1), y(1), 'marker', 'o', 'color', [0 0.5 0]);
    %
    plot(x(end), y(end), 'marker', 's', 'color', [0.5 0 0]);

```

```
170 % hold off
    % subplot(221),ylabel('FL (AU)'), set(gca, 'box','on', 'xlim',ax(1:2))
    % subplot(222),ylabel('Area (Pixels)'), set(gca, 'box','on', 'xlim',ax(1:2))
    % subplot(223),ylabel('C_V (AU)'), set(gca, 'box','on', 'xlim',ax(1:2))
    % subplot(224),ylabel('[GLC] (%w/v)'), set(gca, 'box','on')
```

B.10 Complete source code for segmentor modules

```

function [seg, L, LW, J, K, H] = imsegbact(I, seg, varargin)
opts = getopts(varargin);

nThresh      = parseopts('thresh', opts, 0.4);
5 iSmooth     = parseopts('smoothkernels', opts, [3, 15]);
nEccFlt      = parseopts('eccentricity', opts, [0 inf]);
nAreaFlt     = parseopts('area', opts, [0 inf]);
nSolidFlt    = parseopts('solidity', opts, [0 1]);
nIntsFlt     = parseopts('intensity', opts, [0 mean(I(:))*1.1]);
10
segprev      = parseopts('mergeprev', opts, []);

J = imfilter(imcomplement(I), fspecial('average', iSmooth(1)), 'replicate');
K = imfilter(imcomplement(I), fspecial('average', iSmooth(2)), 'replicate');
15 H = double(imclearborder(imadjust(J-K)));

B = H >= max(H(:))*nThresh;

% first pass watershed to help define separable objects
20 D = bwdist(B);
LW = watershed(D);

% use look-up tables to find edges of objects
lut = makelut('std(x(:)) >= 0.5', 3);
25 C = ~bwmorph(applylut(B, lut), 'skel');
B = B & C;

B = imdilate(B, strel('square', 2));
B(find(LW == 0)) = 0;
30
L = bwlabeln(B);

seg.objallidx = find(bwperim(B));
seg.objqntidx = find(bwperim(imclearborder(B)));
35
% merge the previous segmentation mask with the current on to make editing
% easier and possibly more accurate. shrink the previous mask a little so
% that the new object boundaries are more likely to prevail.
if ~isempty(segprev) & isstruct(segprev),
40     seg.objallidx = find(bwperim(B | bwmorph(bndidx2mask(segprev.imsz, ...
                                                segprev.objallidx), 'erode')));
     seg.objqntidx = find(bwperim(B | bwmorph(bndidx2mask(segprev.imsz, ...
                                                segprev.objqntidx), 'erode')));
end;

```

```

function [seg, L, C, LW] = imsegbf2(I, seg, varargin);
% IMSEGBF
%     [seg, L, B, LW] = imsegbf(I, seg, ...)
%
5 % Segments objects in brightfield the brightfield image I. Objects must
% appear bright with dark borders for this algorithm to work accurately.
% SEG is a structure that contains information regarding the image being
% segmented with the following fields:

```

```

%
10 %         imgid         image id (frame number)
%         locid          scan location id
%         channel        channel name
%         roi            region of interest vector (see IMCROP)
%         imsize         size of the (cropped) image to be segmented
15 %
% When the algorithm returns, the following fields will be appended to
% SEG:
%
%         objqntidx      indices of object borders to quantify
20 %         objallidx     indices of all object borders
%
% The algorithm returns:
%
%         L             label matrix of segmented objects
25 %         B             binary object mask of ALL objects (incl. objs on img edge)
%         LW            watershed region label matrix
%         seg           segmentation data
%
% Allows for manual correction of segmentation if 'autoonly' parameter is
30 % set to FALSE.
%
% Optional Arguments
%         maplev          default: 9000
%         intensity level in first pass marker set to threshold for
35 %         object border definition.
%
%         threshlev      default: 0.7*2^16
%         intensity level in second pass marker processing to threshold
%         for object border definition.
40 %
%         areacutoff     default: [0 Inf]
%         1x2 vector specifying the size filtration limits (number of
%         pixels) that define desired objects. The range is inclusive at
%         both ends -- e.g if the value is set to [25 500], only objects
45 %         with pixel areas from 25 to 500 are retained.
%
%         autoonly       default: true
%         specifies that the algorithm operate completely automatically.
%         Set to FALSE if you wish to view and manually correct
50 %         segmentation. If set to FALSE you must also provide 'SEGDATA'.
%
% History
%         2006/02/20, WLP: Created.
%         2006/03/01, WLP: Incorporated SEGEDIT gui as the primary means
55 %         of editing segmentation masks.
%         2006/03/10, WLP: Changed primary segmentation algorithm to
%         closing by reconstruction. Old image contrast
%         enhancement method kept for reference.
%         2006/03/20, WLP: Removed manual segmentation portion. More
60 %         efficient for users must edit after auto
%         segmentation is complete. Command line
%         arguments and output for SEGEDIT removed
%         removed as well
%
65

```

```

opts          = getopt(varargin);

% these were options for the older segmentation code using image intensity
% and variance
70 iAreaFilter  = parseopts('areacutoff', opts, [0 Inf]);
   iAreaLo     = iAreaFilter(1);
   iAreaHi     = iAreaFilter(2);
   nThreshLev  = parseopts('threshlev', opts, 0.01);
   nBlackLev   = parseopts('blacklev', opts, 3.5);
75 % -----

% new/only option for the new segmentation code using morph reconstruction
iObjRadius    = parseopts('objradius', opts, [2, 15]);
% -----
80
bAutoOnly     = parseopts('autoonly', opts, true);
segprev       = parseopts('mergprev', opts, []);

% emphasize the dark borders that appear on the outer edges of cells
85 cl = class(I);
   switch cl,
       case {'uint8', 'uint16'},
           MAX = intmax(cl);
       case 'double',
90         MAX = max(I(:));
       case 'logical',
           MAX = 1;
       otherwise,
           end;
95
   mn = double(min(I(:)))/MAX;
   st = double(std(I(:)))/MAX;
   mx = double(max(I(:)))/MAX;

100 Iori = I; % save the original image for manual segmentation editing

bExpt = true;
if bExpt,
    % create morphological reconstructions of the source image to remove
105 % high frequency noise 'objects'. Use two reconstruction kernels, one
    % to smooth out internal cell features, the other (larger) to smooth
    % out the image in general.
    Ie = imerode(I, strel('disk', min(iObjRadius)*2));
        % originally square, 9
110 Iers = imreconstruct(Ie, I);

    Ie = imerode(I, strel('disk', max(iObjRadius)*2));
        % originally square, 25
    Ierl = imreconstruct(Ie, I);
115
    % enhance the edges of objects by removing internal features from
    % objects. find these edges by image variance and enhance using
    % simple morphological closing and median filtering
    J = imadjust(imsubtract(imcomplement(Iers), Ierl));
120 S = imadjust(uint16(normrange(stdfilt(J))*2^15*1.5));
    S = imclose(S, strel('square', 3));
    S = medfilt2(S, [3 3]);

```

```

% thicken edges using a 4-con averaging filter
125 C = imfilter(S, [0 1 0;1 1 1; 0 1 0], 'corr', 'replicate');

% segment object borders
B = im2bw(C, graythresh(C));
% reduce borders to thin segments
130 B = bwmorph(B, 'thin', 5);
% remove dangling segments
B = bwmorph(B, 'spur', 5);
% remove isolated pixels
B = bwmorph(B, 'clean');
135 % thicken borders
B = imdilate(B, strel('disk', 1));
% create object seeds
B = imclearborder(~B);
% remove large objects (max obj radius is 15)
140 B = imtophat(B, strel('disk', max(iObjRadius)));
% remove small objects (min obj radius is 2) and smooth object edges
% last two steps function as an area filter
B = imopen(B, strel('disk', min(iObjRadius)));

145 else,
% before any processing get the image variance. this will help to
% remove spurious objects
V = imfill(uint16(image_variance(I, 3)), 'holes');
VD = imdilate(V, strel('disk', 3));
150 VCR = imreconstruct(V, VD);
BV = im2bw(VCR, graythresh(VCR));

I = imadjust(I, [mn*nBlackLev mx], [], 1);

155 I1 = imadjust(medfilt2(imfilter(imcomplement(I), ...
                                fspecial('disk', 3), 'replicate'), [3 3]));
I2 = imadjust(medfilt2(imfilter(I, ...
                                fspecial('disk', 3), 'replicate'), [3 3]));
I2 = imfill(I2, 'holes');

160 I = normrange(double(imsubtract(I1, I2)));
I = imadjust(I, [0.3 0.7], []);

J = I;

165 % create an initial outline mask by thresholding to create white pixels
% at the edges of objects
B = J > nThreshLev;

170 % find initial object boundaries, this will help to keep valid objects
% that would be removed in the later segmentation steps
C = imclose(B, strel('disk', 8));
BND = bwboundaries(C);
BC = zeros(size(B));
175 for i = 1:length(BND),
    BC(sub2ind(size(B), BND{i}(:, 1), BND{i}(:, 2))) = 1;
end;
B = B | BC;

```

```

180 % remove large dark regions and regions that touch the image boundary
    B = ~imclearborder(~B);

    % bridge some of the gaps in the object boundaries
    % flip the intensity to get the initial seed mask
185 B = ~imclose(B, strel('disk', 3));

    % area filter, remove small objects above and below threshold range
    B = bwareaopen(B, iAreaLo) & ~bwareaopen(B, iAreaHi);

190 % variance filter, remove objects that are not in the object map
    % generated by the image variance
    B = B & BV;

end; % if ~bExpt
195

    % create object markers using a distance transform and segment with
    % watershed to generate initial dividing borders
    D = bwdist(B);
200 C = imerode(imregionalmin(D), strel('disk', 3));
    E = bwdist(C);

    LW = watershed(E);
    damidx = find(LW == 0);
205

    % thicken object markers to ensure complete segmentation
    B = bwmorph(B, 'thicken', 3);

    % enforce object boundaries using watershed lines
210 B(damidx) = 0;

    % All objects
    C = B;

215 seg.objallidx = find(bwperim(C));
    seg.objqntidx = find(bwperim(imclearborder(C)));

    % merge the previous segmentation mask with the current on to make editing
    % easier and possibly more accurate. shrink the previous mask a little so
220 % that the new object boundaries are more likely to prevail.
    if ~isempty(segprev) & isstruct(segprev),
        seg.objallidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
            segprev.objallidx), 'erode')));
        seg.objqntidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
225 segprev.objqntidx), 'erode')));
    end;

    B = imclearborder(B);
    L = bwlabeln(B);

```

```

_____ imsegbf3.m _____
function [seg, L, C, W] = imsegbf3(I, seg, varargin)
opts = getopts(varargin);
segprev = parseopts('mergprev', opts, []);
pctSat = parseopts('pctsat', opts, 0.05);

```



```

bDebug          = parseopts('debug', opts, false);
bGraphic        = parseopts('graphic', opts, false);

% test images:
10 % I = imread('H:\05112006\expt-01\brite\brite_000_0799','tiff') - 2^15;
% I = imread('H:\05112006\expt-01\brite\brite_000_0000','tiff') - 2^15;
% I = imread('H:\05112006\expt-01\brite\brite_000_0411','tiff') - 2^15;
% I = imread('H:\01132006\expt-01\brite\brite_000_0311','tiff') - 2^15;
% I = imread('H:\02272006\expt-01\brite\brite_000_0234','tiff') - 2^15;
15
% =====
% CODE BEGINS HERE
%
Z = false(size(I));
20
% contrast enhance the image by saturation
if bGraphic,
    figure, imshow(I, []), set(gcf, 'name', 'Original Image');
    set(gcf, 'units', 'pixels', 'position', [50 50 fliplr(size(I))]);
25    set(gca, 'position', [0 0 1 1]);
end;

I = imsaturate(I, pctSat, varargin);

30 if bGraphic,
    figure, imshow(I, []), set(gcf, 'name', 'Contrast Enhanced Image');
    set(gcf, 'units', 'pixels', 'position', [50 50 fliplr(size(I))]);
    set(gca, 'position', [0 0 1 1]);
end;
35
% threshold
iStats = [mean(double(I(2:end-1))) std(double(I(2:end-1)))];
B = I < (iStats(1) - iStats(2)/2);

40 % morph ops to remove vacuolar objects
B = bwmorph(B, 'skel', inf);
B = bwmorph(B, 'spur', 10);
B = bwmorph(B, 'clean');

45 % segment the objects
B = bwmorph(B, 'dilate');
B = imcomplement(B);
B = imclearborder(B);
B = imfill(B, 'holes');
50
% round out the edges and remove some of the smaller objects
B = imopen(B, strel('square', 5)); % initial object markers

if bGraphic,
55    figure, imshow(B), set(gcf, 'name', 'Initial Markers')
    set(gcf, 'units', 'pixels', 'position', [50 50 fliplr(size(I))]);
    set(gca, 'position', [0 0 1 1]);
end;

60 % to aid in the separation of closely spaced objects find the distance
% transform peaks and mark.
C = imregionalmax(bwdist(~B));

```

```

B2 = imdilate(bwmorph(B & ~imdilate(C, true(3)), 'thin', inf), true(2));
B2 = imclearborder(~B2, 4);
65
% perform watershed using new marker set and split blob objects in original
% marker set using segmented watershed dams
B2 = imclearborder(~bwmorph(xor(B, B2), 'thin', 5), 4);
D = bwdist(B2, 'euclid');
70
if bGraphic,
    figure, imshow(B2), set(gcf, 'name', 'Centroid Enforced Markers')
    set(gcf, 'units', 'pixels', 'position', [50 50 fliplr(size(I)]]);
    set(gca, 'position', [0 0 1 1]);
75 end;

L = watershed(D, 4);

% get the watershed dams
80 W = Z;
W(L == 0) = 1;
W = bwmorph(W, 'thin', inf);
BDR = imdilate(W, strel('square', 2));

85 if bGraphic,
    figure, imshow(I), hold on
    Lrgb = label2rgb(bwlabeln(B & ~BDR), 'jet', 'k', 'shuffle');
    hMkr = imshow(Lrgb);
    hold on
90    Wrgb = double(cat(3, BDR, BDR, Z));
    hBdr = imshow(Wrgb);

    AMkr = double((B & ~BDR))*0.5;% + double(~(B & ~BDR));
    ABdr = double(BDR);
95
    set(hMkr, 'AlphaData', AMkr);
    set(hBdr, 'AlphaData', ABdr);
    set(gcf, 'name', 'Segmentation Overlay')
    set(gcf, 'units', 'pixels', 'position', [50 50 fliplr(size(I)]]);
100    set(gca, 'position', [0 0 1 1]);
    end;

L = bwlabeln(B & ~BDR);
C = L > 0;
105
seg.objallidx = find(bwperim(C));
seg.objqntidx = find(bwperim(imclearborder(C)));

% merge the previous segmentation mask with the current on to make editing
110 % easier and possibly more accurate. shrink the previous mask a little so
% that the new object boundaries are more likely to prevail.
if ~isempty(segprev) && isstruct(segprev),
    seg.objallidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
        segprev.objallidx), 'erode')));
115    seg.objqntidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
        segprev.objqntidx), 'erode')));
end;

% if bGraphic,

```

```

120 %     figure, imshow(I, []), hold on
%     hImDams = imshow(double(cat(3, W, W, Z)));
%     set(hImDams, 'AlphaData', 0.3);
%     hImLabels = imshow(label2rgb(L, 'jet', 'k', 'shuffle'));
%     set(hImLabels, 'AlphaData', 0.3);
125 % end;

```

```

_____ imsegbf4.m _____
function [seg, L, C, W] = imsegbf4(I, seg, varargin)
% segmentation combining the best of bf2 and bf3 algorithms

opts           = getoptns(varargin);
5 segprev      = parseopts('mergeprev', opts, []);

bDebug         = parseopts('debug', opts, false);
bGraphic       = parseopts('graphic', opts, false);
nGraphicPos    = parseopts('position', opts, [50 50 fliplr(size(I))]);
10
% new/only option for the new segmentation code using morph reconstruction
iObjRadius     = parseopts('objradius', opts, [2, 15]);

% saturation level for contrast enhancement
15 pctSat      = parseopts('pctsat', opts, 0.05);

% option to use convex hulls of objects, helps to reduce oversegmentation
% but might produce under segmented results
bConvexObjs    = parseopts('convexobjs', opts, false);
20
Z = false(size(I));

% contrast enhance the image by saturation
if bGraphic,
25     figure, imshow(I, []), set(gcf, 'name', 'Original Image');
        set(gcf, 'units', 'pixels', 'position', nGraphicPos);
        set(gca, 'position', [0 0 1 1]);
    end;

30 I = imsaturate(I, pctSat, varargin);

    if bGraphic,
        figure, imshow(I, []), set(gcf, 'name', 'Contrast Enhanced Image');
        set(gcf, 'units', 'pixels', 'position', nGraphicPos);
35     set(gca, 'position', [0 0 1 1]);
    end;

% create morphological reconstructions of the source image to remove
% high frequency noise 'objects'. Use two reconstruction kernels, one
40 % to smooth out internal cell features, the other (larger) to smooth
% out the image in general.
Ie = imerode(I, strel('disk', min(iObjRadius)*2)); % originally square, 9
Iers = imreconstruct(Ie, I);

45 Ie = imerode(I, strel('disk', max(iObjRadius)*2)); % originally square, 25
Ierl = imreconstruct(Ie, I);

% if bGraphic,
%     figure, imshow(Iers, []), set(gcf, 'name', 'Small Open by Reconstruction');

```

```

50 %     set(gcf, 'units', 'pixels', 'position', nGraphicPos);
%     set(gca, 'position', [0 0 1 1]);
%
%     figure, imshow(Ierl, []), set(gcf, 'name', 'Large Open by Reconstruction');
%     set(gcf, 'units', 'pixels', 'position', nGraphicPos);
55 %     set(gca, 'position', [0 0 1 1]);
% end;

% enhance the edges of objects by removing internal features from
% objects. find these edges by image variance and enhance using
60 % simple morphological closing and median filtering
J = imadjust(imsubtract(imcomplement(Iers), Ierl));
S = imadjust(uint16(normrange(stdfilt(J))*2^15*1.5));
S = imclose(S, strel('square', 3));
S = medfilt2(S, [3 3]);
65
% thicken edges using a 4-con averaging filter
% C = imfilter(S, [0 1 0; 1 1 1; 0 1 0], 'corr', 'replicate');

if bGraphic,
70     figure, imshow(J, []), set(gcf, 'name', 'Object Edge Detection');
        set(gcf, 'units', 'pixels', 'position', nGraphicPos);
        set(gca, 'position', [0 0 1 1]);

        figure, imshow(S, []), set(gcf, 'name', 'Object Edge Enhancement');
75     set(gcf, 'units', 'pixels', 'position', nGraphicPos);
        set(gca, 'position', [0 0 1 1]);
end;

iStats = [mean(double(I(2:end-1))) std(double(I(2:end-1)))];
80
% threshold high contrast borders and edges detected by local variance
B = (I < (iStats(1) - iStats(2)/2));
% B = bwmorph(B, 'thin', 10);
% B = bwmorph(B, 'spur', max(iObjRadius)*2);
85 % B = bwmorph(B, 'clean');
% B = bwmorph(B, 'dilate');
% B = imclearborder(~B);
% B = imfill(B, 'holes');
% B1 = B;
90
B = B | im2bw(S, graythresh(S));
% reduce borders to thin segments
B = bwmorph(B, 'thin', 10);
% remove dangling segments
95 B = bwmorph(B, 'spur', max(iObjRadius)*2);
% remove isolated pixels
B = bwmorph(B, 'clean');
% thicken borders
B = imdilate(B, strel('disk', 1));
100 % create object seeds
B = imclearborder(~B);

B = imfill(B, 'holes');
% B2 = B;
105 %
% B = B1 | B2;

```

```

% remove large objects (max obj radius is 15)
B = imtophat(B, strel('disk', max(iObjRadius)));
% remove small objects (min obj radius is 2) and smooth object edges
110 % last two steps function as an area filter
    B = imopen(B, strel('disk', min(iObjRadius)));

    if bConvexObjs,
        B2 = Z;
115        % use convex hull to 'close' remaining u-bended objects ... might be
        % inefficient. Rebuild the mask using convex hull images, one object
        % at a time.
        for i = 1:2,
            L = bwlabeln(B);
120            stObj = regionprops(L, 'conveximage', 'pixellist');
            for iCurrObj = 1:length(stObj),
                pxlst = stObj(iCurrObj).PixelList;
                bbmin = min(pxlst);
                bbmax = max(pxlst);
125
                cim = stObj(iCurrObj).ConvexImage;

                ii = bbmin(2);
                ij = bbmax(2);
130                ji = bbmin(1);
                jj = bbmax(1);

                B2(ii:ij, ji:jj) = B2(ii:ij, ji:jj) | cim;
            end;
135            B = B | B2;
        end;
    end;

    if bGraphic,
140        figure, imshow(B), set(gcf, 'name', 'Initial Markers');
        set(gcf, 'units', 'pixels', 'position', nGraphicPos);
        set(gca, 'position', [0 0 1 1]);
    end;

145 % to aid in the separation of closely spaced objects find the distance
    % transform peaks and mark.
    C = imregionalmax(bwdist(~B));
    B2 = imdilate(bwmorph(B & ~imdilate(C, true(min(iObjRadius)*2)), ...
        'thin', inf), true(2));
150 B2 = imclearborder(~B2, 4);

    % perform watershed using new marker set and split blob objects in original
    % marker set using segmented watershed dams
    B2 = imclearborder(~bwmorph(xor(B, B2), 'thin', min(iObjRadius)), 4);
155 D = bwdist(B2, 'euclid');

    if bGraphic,
        figure, imshow(B2), set(gcf, 'name', 'Centroid Enforced Markers')
        set(gcf, 'units', 'pixels', 'position', nGraphicPos);
160        set(gca, 'position', [0 0 1 1]);
    end;

    L = watershed(D, 4);

```

```

165 % get the watershed dams
    W = Z;
    W(L == 0) = 1;
    W = bwmorph(W, 'thin', inf);
    BDR = imdilate(W, strel('square', 2));
170
    if bGraphic,
        figure, imshow(I), hold on
        Lrgb = label2rgb(bwlabeln(B & ~BDR), 'jet', 'k', 'shuffle');
        hMkr = imshow(Lrgb);
175    hold on
        Wrgb = double(cat(3, BDR, BDR, Z));
        hBdr = imshow(Wrgb);

        AMkr = double((B & ~BDR))*0.5;
180    ABdr = double(BDR);

        set(hMkr, 'AlphaData', AMkr);
        set(hBdr, 'AlphaData', ABdr);
        set(gcf, 'name', 'Segmentation Overlay')
185    set(gcf, 'units', 'pixels', 'position', nGraphicPos);
        set(gca, 'position', [0 0 1 1]);
    end;

    L = bwlabeln(B & ~BDR);
190 C = L > 0;

    seg.objallidx = find(bwperim(C));
    seg.objqntidx = find(bwperim(imclearborder(C)));

195 % merge the previous segmentation mask with the current on to make editing
% easier and possibly more accurate. shrink the previous mask a little so
% that the new object boundaries are more likely to prevail.
    if ~isempty(segprev) && isstruct(segprev),
        seg.objallidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
200         segprev.objallidx), 'erode')));
        seg.objqntidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
         segprev.objqntidx), 'erode')));
    end;

```

```

_____ imsegred.m _____
function [seg, L, LW, cfiltervals] = imsegred(I, seg, varargin)
% IMSEGRED
% [seg, L, LW, cfiltervals] = imsegred(I, seg, ...)
%
5 % Description
% Segments circular/elliptical objects that appear dark against a light
% background. Designed to find yeast cells imaged in a monolayer, backlit
% by red fluorescence, but may work on images containing similar features.
%
10 % Returns the label matrices L and LW for segmented objects and watershed
% regions respectively, and updates the segmentation data the structure
% seg. To generate a generic binary mask use,
%
% B = (L > 0);
15 %

```

```

% To extract watershed catchment basin boundaries, use,
%
%   W = (LW == 0);
%
20 % If no additional filtration constraints are specified, defaults are used,
%   which will include all segmented objects. Filterable values for each
%   object is returned in the cell array cfiltervals containing vectors for:
%
%   {perimeter, area, shape factor (circularity), eccentricity, solidity}
25 %
% Optional parameters and filtration constraints are specified in
% ('parameter', value) pairs and are as follows:
%
%   thresh          default: 0.4
30 %       Percent of contrast enhanced range to segment initial seeds.
%       Should be greater than 0 and no larger than 1.
%
%   smoothkernels   default: [3, 15]
%       Small and large gaussian smoothing kernel size used to extract
35 %       local maxima from the image for object marking.
%
%   area            default: [0, inf]
%       Object area threshold range, inclusive. Objects whose pixel area
%       fall out side of this range are removed from the returned label
40 %       matrices.
%
%   eccentricity    default: [0, inf]
%       Inclusive range. Object region property (See REGIONPROPS). A
%       perfect circle will have an eccentricity of 0, where a line segment
45 %       will have a value of 1.
%
%   solidity        default: [0, 1]
%       Inclusive range. Object region property (See REGIONPROPS).
%       Proportion of the pixels in the region's convex hull that also
50 %       exist in the region area. A highly contorted region will have a
%       solidity that tends toward 0, where a convex object (e.g. a circle
%       or an ellipse) will a solidity value at or near 1.
%
%   intensity       default: [0 <10% of image mean>]
55 %       Inclusive range. Removes objects whos mean image intensity do not
%       fall within the specified range.
%
% Example:
%   Return objects having eccentricity values >= 0.9 and intensity values
60 %   one standard deviation below the image maximum:
%
%       [L, LW] = imsegred(I, seg, 'eccentricity', [0.9 inf], ...
%                               'intensity', [0 max(I(:))-std(I(:))])
%
65 % History:
%   2006/02/06, WLP:   Created by Wyming Lee Pang, SBL, UCSD.
%   2006/03/08, WLP:   Descriptive header added. Removed binary mask
%                       return since it can easily be regenerated from the
70 %                       label matix L.
%   2006/03/10, WLP:   Added compatibility with new segmentation data
%                       storage method.

```

```

    opts = getoptopts(varargin);
75
    nThresh      = parseopts('thresh', opts, 0.4);
    iSmooth      = parseopts('smoothkernels', opts, [3, 15]);
    nEccFlt      = parseopts('eccentricity', opts, [0 inf]);
    nAreaFlt     = parseopts('area', opts, [0 inf]);
80 nSolidFlt    = parseopts('solidity', opts, [0 1]);
    nIntsFlt     = parseopts('intensity', opts, [0 mean(I(:))*1.1]);

    segprev     = parseopts('mergeprev', opts, []);

85 % identify local regional maxima
    J = imfilter(imcomplement(I), ...
                fspecial('gaussian', iSmooth(1), iSmooth(1)), 'replicate');
    K = imfilter(imcomplement(I), ...
                fspecial('gaussian', iSmooth(2), iSmooth(2)), 'replicate');
90 H = double(imclearborder(imadjust(J-K)));

    % threshold and fill holes to generate initial marker seeds
    B = H >= max(H(:))*nThresh;
    B = imfill(B, 'holes');
95

    % first pass watershed to help define separable objects
    D = bwdist(B);
    LW = watershed(D);

100 B = imdilate(B, strel('square', 2));
    B(find(LW == 0)) = 0;

    L = bwlabeln(B);

105 % get object properties and filter based on eccentricity, area, and
    % solidity and rebuild the mask accordingly
    C = zeros(size(L));
    S = regionprops(L, 'image', 'pixelidylist', 'eccentricity', 'solidity');
    for i=1:length(S),
110     oCurrObj = S(i);

        v(i) = mean(mean(I(oCurrObj.PixelIdxList))); % object intensity
        p(i) = sum(sum(bwperim(oCurrObj.Image))); % object perimeter
        a(i) = sum(sum(oCurrObj.Image)); % object area
115     f(i) = p(i)^2/4/pi/a(i); % shape factor 1 -> circle
        e(i) = oCurrObj.Eccentricity;
        s(i) = oCurrObj.Solidity;

120     bIsValidEcc = (e(i) >= nEccFlt(1) & e(i) <= nEccFlt(2));
        bIsValidArea = (a(i) >= nAreaFlt(1) & a(i) <= nAreaFlt(2));
        bIsValidSolid = (s(i) >= nSolidFlt(1) & s(i) <= nSolidFlt(2));
        bIsValidInts = (v(i) >= nIntsFlt(1) & v(i) <= nIntsFlt(2));

125     bIsValid = bIsValidEcc & bIsValidArea ...
                & bIsValidSolid & bIsValidInts;

        C(oCurrObj.PixelIdxList) = bIsValid;
    end;

```



```
130 % reperform the watershed segmentation on the filtered mask
    D = bwdist(C);
    LW = watershed(D);
    C(find(LW == 0)) = 0;
135 B = C;
    L = bwlabeln(B);

    cfiltervals = {p, a, f, e, s};

140 % output segmentation data.
    seg.objallidx = find(bwperim(C));
    seg.objqntidx = find(bwperim(imclearborder(C)));

    % merge the previous segmentation mask with the current on to make editing
145 % easier and possibly more accurate. shrink the previous mask a little so
    % that the new object boundaries are more likely to prevail.
    if ~isempty(segprev) & isstruct(segprev),
        seg.objallidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
            segprev.objallidx), 'erode')));
150     seg.objqntidx = find(bwperim(C | bwmorph(bndidx2mask(segprev.imsz, ...
            segprev.objqntidx), 'erode')));
    end;
```

B.11 Complete source code for library functions

```

_____ bndidx2mask.m _____
function msk = bndidx2mask(sz, idx)
% converts boundary pixels indices to a binary mask for object segmentation

msk = zeros(sz);
5 msk(idx) = 1;
msk = imfill(msk, 'holes');
_____

_____ caf.m _____
closeallfigs
_____

_____ closeallfigs.m _____
% closeallfigs.m
close(get(0,'children'))
_____

_____ createcorrimgs.m _____
function [N, F] = createcorrimgs(sCorrPath, iCorrImgs, ...
                                csCorrNoise, csCorrFlat)
% CREATECORRIMGS
% [N, F] = createcorrimgs(sCorrPath, iCorrImgs, csCorrNoise, csCorrFlat)
5 %
% Creates a set of correction images for use in quantitative fluorescence
% analysis.
%
% sCorrPath Location of correction image channel directories
10 %
% iCorrImgs Vector of image indices to use, e.g. [0:9]. This vector is
% used for all correction images created.
%
% csCorrNoise Qx1 Cell array of noise correction channels to process.
15 % Processed noise correction frames are output to N which is
% a Qx2 cell array, where the first column contains the
% channel name and the second contains the correction frame.
%
% csCorrFlat Qx1 Cell array of flatfield correction channels to process.
20 % Processed flatfield correction frames are output to F which
% is a Qx2 cell array, where the first column contains the
% channel name and the second contains the correction frame.
%
%
25 % HISTORY
% 2006/02/23, WLP: Created.
% 2006/10/01, WLP: Channel names that are 'none' or blank are
% skipped. Before they would crash the program.
% 2006/10/01, WLP: Added function documentation header.
30 %
for i = 1:length(csCorrNoise),
    N{i, 1} = csCorrNoise{i};
    N{i, 2} = [];
    for j = iCorrImgs,
35 % camera bias
        if exist([sCorrPath '/' csCorrNoise{i}], 'dir') ...
            && (~strcmpi(csCorrNoise{i}, 'none') ...
                || ~isempty(strtrim(csCorrNoise{i}))),
% printf('Reading: %s, %04d', csCorrNoise{i}, j);

```

```

40         N{i, 2} = cat(3, N{i, 2}, ...
            getimg(csCorrNoise{i}, 0, j, 'basepath', sCorrPath) - 2^15);
        end;
    end;

45    if ~isempty(N{i, 2}),
        N{i, 2} = mean(double(N{i, 2}), 3);
    else
        N{i, 2} = 0;
    end;
50 end;

    for i = 1:length(csCorrFlat),
        F{i, 1} = csCorrFlat{i};
        F{i, 2} = [];
55    for j = iCorrImgs,
        % image flatfields
        if exist([sCorrPath '/' csCorrFlat{i}], 'dir') ...
            && (~strcmpi(csCorrFlat{i}, 'none') ...
                || ~isempty(strtrim(csCorrFlat{i}))),
60 %         printf('Reading: %s, %04d', csCorrFlat{i}, j);
            F{i, 2} = cat(3, F{i, 2}, ...
                getimg(csCorrFlat{i}, 0, j, 'basepath', sCorrPath) - 2^15);
        end;
    end;

65    if ~isempty(F{i, 2}),
        F{i, 2} = mean(double(F{i, 2}), 3);
        F{i, 2} = F{i, 2}./max(F{i, 2}(:));
    else
70        F{i, 2} = 1;
    end;
end;

```

```

_____ getching.m _____
function I = getching(sChannel, cIall)
I = [cIall{strmatch(sChannel, {cIall{:}, 1}}, 'exact'), 2}];

```

```

_____ getdatacol.m _____
function [datavec] = getdatacol(sCol, Data, varargin)
%GETDATACOL
% [datavec] = getdatacol(sCol, Data, cols)
%
5 % Gets data from a column in a data set
if nargin == 3,
    cols = varargin{1};
elseif nargin == 2,
    if isstruct(Data),
10        cols = Data.colheaders;
        Data = Data.data;
    else,
        error(['invalid input: '...
            'Data must be a structure or column '...
15        'headers must be defined']);
    end;
end;

datavec = Data(:, strmatch(upper(sCol), strtrim(upper(cols)), 'exact'));

```

```

function [I Iimf] = getimg(sChannel, iLocID, iImgID, varargin)
%GETIMG
% [I Iimf] = getimg(sChannel, iLocID, iImgID, [options])
%
% Retrieves image pixel and file information data.
%
% Options include 'basepath' and 'imgfmt'.
%
% Values of 'imgfmt' are the same as the format specifier for imread()
10 % and imfinfo(). Default format is TIF.
%
% For 'basepath', usage order is user specified path, the global variable
% g_sBasePath, and current directory '.' as base path.
%
% Assumes images are organized:
15 % <base path>/<channel>/<channel>_<locid:03>_<imgid:04>.<fmt>

global g_sBasePath

20 opts = getopt(s(varargin);
    sFileFmt = parseopts('imgfmt', opts, 'tif');
    sCurrBasePath = parseopts('basepath', opts, '');

    if ~isempty(g_sBasePath) & strcmp(sCurrBasePath, ''),
25     sCurrBasePath = g_sBasePath;
    end;

    if strcmp(sCurrBasePath, ''),
        sCurrBasePath = '.';
30 end;

    sImgFile = sprintf('%s/%s/%s_%03d_%04d', sCurrBasePath, sChannel, ...
35     sChannel, iLocID, iImgID);

    I = imread(sImgFile, sFileFmt);

    if nargin > 1,
40     Iimf = imfinfo(sImgFile, sFileFmt);
    end;

```

```

function opts = getopt(s(varargin)
% GETOPTS
% opts = getopt(...);
%
% Gets options as param, value pairs from a varargin cell array.
% Intended use for custom functions.
% Output 'opts' is a cell array where opts(:,1) is a list of all the
% options names provided and opts(:,2) are their corresponding values.
%
10 % If no options are specified, returns and empty array.

```

```

opts = [];
args = varargin{:};
if ~isempty(args),
15   % there are options
       if mod(length(args),2) == 0,
           optn = 0;
           for iOpt = 1:2:length(args),
               optn = optn + 1;
20           param = lower(args{iOpt});
               value = args{iOpt+1};
               opts{optn, 1} = param;
               opts{optn, 2} = value;
           end;
25   end;
end;

```

```

_____ hashval.m _____
function val = hashval(cvHash, sKey)
val = cvHash(strmatch(sKey, cvHash(:, 1), 'exact'), 2:end);

```

```

_____ imdispseg.m _____
function imdispseg(Iseg, L, varargin)
opts = getopts(varargin);
sMethod = upper(parseopts('method', opts, 'alpha'));

5 switch sMethod,
    case 'ALPHA',
        % displays segmented object labels as a semi-transparent overlay
        % over the image used for segmentation

10     Lrgb = label2rgb(L, 'jet', 'w', 'shuffle');
        imshow(Iseg, []), hold on
        hImsk = imshow(Lrgb);
        set(hImsk, 'AlphaData', 0.3);
        hold off

15     case 'BORDER',
        % this creates a border around segmented objects rather than an
        % alpha transparency overlay. use this if operating via remote
        % desktop. this also might be faster.

20         Imsk = zeros(size(L));
        Imsk(find(L)) = 1;
        Imsk = bwperim(Imsk);
        Lrgb = double(label2rgb(bwlabeln(Imsk), 'jet', 'k', 'shuffle'))/255;

25         Idsp = double(Iseg);
        Idsp = (Idsp-min(Idsp(:)))/(max(Idsp(:)) - min(Idsp(:)));

        Ir = Idsp;
30         Ig = Idsp;
        Ig(find(Imsk)) = 1;
        Ib = Idsp;

        RGB = cat(3, Ir, Ig, Ib) + Lrgb;

35         %imshow(label2rgb(L), []), colormap prism

```

```

        imshow(RGB)

        otherwise,
40         % nothing
        end;

```

```

_____  imsaturate.m _____
function [I] = imsaturate(I, pctSat, varargin)
opts      = getopts(varargin);
bDebug    = parseopts('debug', opts, false);

5 iClsRange = getrangefromclass(I);

nPixels = numel(I);
nMinInt = min(double(I(:)));
nMaxInt = max(double(I(:)));
10
% rough estimate of saturation levels
nIntLev = linspace(nMinInt, nMaxInt, 10);
nSatLev = zeros(10,2);
for i = 1:length(nIntLev),
15     nSatLev(i, 1) = sum(I(:) <= round(nIntLev(i)))/nPixels;
        nSatLev(i, 2) = sum(I(:) >= round(nIntLev(i)))/nPixels;
end;

nLoSatRange = [find(nSatLev(:, 1) < pctSat, 1, 'last'), ...
20             find(nSatLev(:, 1) > pctSat, 1, 'first')];
nLoSatRange = round(nIntLev(nLoSatRange));

nHiSatRange = [find(nSatLev(:, 2) > pctSat, 1, 'last'), ...
               find(nSatLev(:, 2) < pctSat, 1, 'first')];
25 nHiSatRange = round(nIntLev(nHiSatRange));

% refine saturation search ranges
% low range
nIntLev = linspace(nLoSatRange(1), nLoSatRange(2), 10);
30 nSatLev = zeros(10, 1);
for i = 1:length(nIntLev),
        nSatLev(i, 1) = sum(I(:) <= round(nIntLev(i)))/nPixels;
end;

35 nLoSatRange = [find(nSatLev(:, 1) < pctSat, 1, 'last'), ...
                 find(nSatLev(:, 1) > pctSat, 1, 'first')];
nLoSatRange = round(nIntLev(nLoSatRange));

% high range
40 nIntLev = linspace(nHiSatRange(1), nHiSatRange(2), 10);
nSatLev = zeros(10, 1);
for i = 1:length(nIntLev),
        nSatLev(i, 1) = sum(I(:) >= round(nIntLev(i)))/nPixels;
end;
45
nHiSatRange = [find(nSatLev(:, 1) > pctSat, 1, 'last'), ...
               find(nSatLev(:, 1) < pctSat, 1, 'first')];
nHiSatRange = round(nIntLev(nHiSatRange));

50 if bDebug,

```

```

        printf('Saturation Ranges:');
        printf('%d\t', nLoSatRange);
        printf('%d\t', nHiSatRange);
    end;
55
    % Adjust the image level according to the lo/hi ranges found
    for i = nLoSatRange(1):1:nLoSatRange(2),
        pctSatCurr = sum(I(:) <= i)/nPixels;
        if pctSatCurr >= pctSat,
60            break;
        end;
    end;
    LoRng = i/max(iClsRange);

65 for i = nHiSatRange(2):-1:nHiSatRange(1),
        pctSatCurr = sum(I(:) >= i)/nPixels;
        if pctSatCurr >= pctSat,
            break;
        end;
70 end;
    HiRng = i/max(iClsRange);

    I = imadjust(I, [LoRng HiRng], []);

```

```

_____ imseg_getobjdata.m _____
function [nObjData, cols] = imseg_getobjdata(csIqnt, seg, varargin)
opts = getopts(varargin);

% csIqnt is a cell array {channel name, <imagedata>} for each channel to
5 % quantify on each row.

% seg is the segmentation data in a structure with the following fields
% imgid, locid, channel, roi, imsize, objqntidx, objallidx

10 % B is the mask for all objects, including those that lie on the image
% borders. L is the label matrix for objects that should be quantified. B
% is used to determine the image background level.
B = bndidx2mask(seg.imsz, seg.objallidx);
L = bwlabeln(imclearborder(B));
15
iImgID = seg.imgid;
iLocID = seg.locid;

csObjProps = {'area', 'centroid', 'pixelidxlabel', 'boundingbox', ...
20             'conveximage', 'filledimage', 'image'};
stObjData = regionprops(L, csObjProps);

% seg.imsz
% csIqnt
25 %
% nBGLevel = [];
nChannels = size(csIqnt, 1);
for i = 1:nChannels,

30     % resize the quantification image to match the segmentation image
    Iqnt = csIqnt{i,2};
    m = mean(size(Iqnt)./seg.imsz);

```

```

    if m ~= 1,
        Iqnt = imresize(Iqnt, m);
35    end;

    nBGLevel(i) = mean2(Iqnt(~B));
end;

40 % loop over each object in the and log values into an array
nObjData = [];
for iObjID=1:length(stObjData),
    stCurrObj = stObjData(iObjID);

45    % get subimage pixels for calculation
    nFL = [];
    for i = 1:nChannels,

        % resize the quantification image to match the segmentation image
50        Iqnt = csIqnt{i,2};
        m = mean(size(Iqnt)./seg.imesize);
        if m ~= 1,
            Iqnt = imresize(Iqnt, m);
        end;

55        % subimage pixels for calculations
        imCurrObj{i} = Iqnt(stCurrObj.PixelIdxList);
        vCurrObj{i} = double(imCurrObj{i}(:));

60        nFL = [nFL, mean(vCurrObj{i}), std(vCurrObj{i}), ...
                min(vCurrObj{i}), max(vCurrObj{i}), nBGLevel(i)];
    end;

    % calculate the object perimeter
65    % there are three types: native, filled, and convex, based on the
    % supplied binary region
    iarPerim = [];
    cimBinReg = {stCurrObj.Image, stCurrObj.FilledImage, ...
                 stCurrObj.ConvexImage};

70    for iBinRegType = 1:length(cimBinReg),
        iarPerim = [iarPerim, sum(sum(bwperim(cimBinReg{iBinRegType})))]];
    end;

    datarow = [...
75        iLocID,...
        iImgID,...
        iObjID,...
        stCurrObj.Centroid,...
        stCurrObj.BoundingBox,...
80        stCurrObj.Area,...
        iarPerim, ...
        nFL ...
        ];

85    nObjData = [nObjData; datarow];
end;

cols = {'LOCID', 'IMGID', 'OBJID', 'CENTX', 'CENTY', 'BBOXX', 'BBOXY', ...
        'BBOXW', 'BBOXH', 'AREA', 'PERIMNATIVE', 'PERIMFILLED', 'PERIMCONVEX'};

```



```

90 for i = 1:nChannels,
    cols = {cols{:}, ...
        sprintf('MEAN%d', i), ...
        sprintf('STD%d', i), ...
        sprintf('MIN%d', i), ...
95     sprintf('MAX%d', i), ...
        sprintf('BGLVL%d', i) ...
    };
end;

```

```

_____ imsegauto.m _____
function imsegauto(runinfo, seginfo, cvChannels, cvROIs)
% stand-alone segmentation code
% uses full auto segmentation routine(s) and stores segmentation data to
% the data directory specified by the user named by img id.
5
% sExptBase      = runinfo.path;
% sExptPrefix    = runinfo.prefix;
% sExptID        = runinfo.id;
% sExptPath      = sprintf('%s%s%s', sExptBase, filesep, ...
10 %                                     sExptPrefix, sExptID);

sExptPath        = runinfo.path;
sCorrPath        = runinfo.corrpath;

15 iLocID         = runinfo.locid;
viIdx            = runinfo.indices;
nTotImgs        = length(viIdx);

sSegDataPath     = [sExptPath, filesep, seginfo.path];
20 bSegOverwrite  = seginfo.overwrite;
sSegAlgo         = seginfo.algorithm;
csSegOptions     = seginfo.opts;
bDispSeg        = seginfo.display;
sDispSegMeth    = seginfo.displaymethod;
25
% read in correction images
[N, F]          = createcorrimgs(sCorrPath, [0:9], ...
                                unique(cvChannels(:, 2)), unique(cvChannels(:, 3)));

30 % create data output directories if needed
if ~exist(sSegDataPath, 'dir'),
    mkdir(sSegDataPath);
end;
if ~exist(sprintf('%s/%03d', sSegDataPath, iLocID), 'dir'),
35     mkdir(sprintf('%s/%03d', sSegDataPath, iLocID));
end;

% preparation complete, start the segmentation process.
hProg = waitbar(0, '');
40 % iProgPos = get(hProg, 'position');
% set(hProg, 'position', [1, 25, iProgPos(3:4)]);

iCurrImg = 0;
tLoopElaps = [];
45 segprev = [];
sFilePath = sprintf('%s%s%03d', sSegDataPath, filesep, iLocID);

```

```

for iCurrIdx = viIdx,
    iCurrImg = iCurrImg + 1;
    nProgress = iCurrImg/nTotImgs;
50
    tLoopInit = clock;
    updateprogress(hProg, iCurrImg, nTotImgs, tLoopElaps);

    % =====
55    % SEGMENTATION CODE
    %

    % check if there is already a segmentation file.  skip the segmentation
    % process if there is.

60
    % get the segmentation data directory listing
    sFileName = sprintf('%04d.mat', iCurrIdx);
    sFileFullPath = sprintf('%s%s%s', sFilePath, filesep, sFileName);

65
    stFiles = dir(sFilePath);
    csFiles = {stFiles(~[stFiles.isdir]).name};

    if ismember(sFileName, csFiles) && ~bSegOverwrite,
        waitbar(nProgress, hProg, 'Reading Existing Segmentation Data');
70        load(sFileFullPath);
        segprev = seg;

        if bDispSeg,
            B = zeros(seg.imsize);
75            B(seg.objallidx) = 1;
            Bbg = B;
            B = imclearborder(B);
            L = bwlabeln(B);
        end;
80    else
        % images only need to be read if new segmentation is to be
        % performed.  this boosts performance.
        waitbar(nProgress, hProg, 'Reading Image Set');
        [cIall, cIseg, cIqnt] = readingset(iLocID, iCurrIdx, sExptPath, ...
85            cvChannels, cvROIs, N, F);

        waitbar(nProgress, hProg, 'Segmenting Objects');

        % initialize the segmentation data structure
90        seg.imgid = iCurrIdx;
        seg.locid = iLocID;
        seg.channel = cIseg{1};

        roi = hashval(cvChannels, seg.channel);
95        roi = hashval(cvROIs, roi{6});
        roi = cell2mat(roi);
        if ~isempty(roi),
            seg.roi = roi;
        else,
100        seg.roi = [0 0 fliplr(size(cIseg{2}))];
        end

        seg.imsize = size(cIseg{2});

```

```

    seg.objqntidx = [];
105    seg.objallidx = [];

    if isempty(cIseg{2}),
        printf('ERROR: Segmentation Image is Empty!');
    end;
110    [seg, L] = feval(sSegAlgo, cIseg{2}, seg, ...
                    {csSegOptions{:}}, ...
                    'autoonly', true, 'mergeprev', segprev));

    segprev = seg;

115    % output segmentation mask data to the specified data directory.
    save(sFileFullPath, 'seg');
end;

120    if bDispSeg,
        if ~exist('hFigSeg'), hFigSeg = figure; end;
        imdispseg(cIseg{2}, L, 'method', sDispSegMeth);
    end;

125    % -----
    % application specific segmentation code here:

    % -----

130    %
    % END SEGMENTATION CODE
    % =====

135    tLoopElaps = [tLoopElaps etime(clock, tLoopInit)];
end;

    if bDispSeg,
        close(hFigSeg);
140    clear hFigSeg
    end;

    close(hProg);
    clear hProg;

```

```

_____ imsegcollect.m _____
function imsegcollect(runinfo, seginfo, quantinfo, cvChannels, cvROIs)
% stand alone script to quantitate image intensity

% Experiment info:
5 % sExptBase      = runinfo.path;
% sExptPrefix    = runinfo.prefix;
% sExptID        = runinfo.id;
% sExptPath      = sprintf('%s%s%s', sExptBase, filesep, ...
%                               sExptPrefix, sExptID);

10 sExptPath      = runinfo.path;
sCorrPath       = runinfo.corrpath;

iLocID          = runinfo.locid;

```

```

15 viIdx          = runinfo.indices;
   nTotImgs      = length(viIdx);

   sSegDataPath  = [sExptPath, filesep, seginfo.path];

20 sObjDataPath   = [sExptPath, filesep, quantinfo.path];
   bVerbose      = quantinfo.verbose;
   bBlobOut      = quantinfo.blobout;
   bOverwrite    = quantinfo.overwrite;

25 % read in correction images
   [N, F]        = createcorrimgs(sCorrPath, [0:9], ...
                               unique(cvChannels(:, 2)), unique(cvChannels(:, 3)));

   % create data output directories if needed
30 if ~exist(sObjDataPath, 'dir'),
       mkdir(sObjDataPath);
   end;
   if ~exist(sprintf('%s/%03d', sObjDataPath, iLocID), 'dir'),
       mkdir(sprintf('%s/%03d', sObjDataPath, iLocID));
35 end;

   % preparation complete, start the collection process.
   hProg = waitbar(0, '');
   % iProgPos = get(hProg, 'position');
40 % set(hProg, 'position', [1, 25, iProgPos(3:4)]);

   iCurrImg = 0;
   tLoopElaps = [];
   segprev = [];
45 for iCurrIdx = viIdx,
       iCurrImg = iCurrImg + 1;
       nProgress = iCurrImg/nTotImgs;

       tLoopInit = clock;
50   updateprogress(hProg, iCurrImg, nTotImgs, tLoopElaps);

       waitbar(nProgress, hProg, 'Reading Image Set');
       [cIall, cIseg, cIqnt] = readimgset(iLocID, iCurrIdx, sExptPath, ...
                                       cvChannels, cvROIs, N, F);

55   waitbar(nProgress, hProg, 'Reading Segmentation Data');
       load(sprintf('%s%s%03d%s%04d.mat', sSegDataPath, filesep, iLocID, ...
                   filesep, iCurrIdx));

60   % =====
   % DATA COLLECTION CODE:
   %
       waitbar(nProgress, hProg, 'Gathering Object Data');

65   [nObjData colsscfl] = imseg_getobjdata(cIqnt, seg);
       sCols = sprintf('%13s', colsscfl{:});
       sData = [];
       for iDataRow = 1:size(nObjData, 1),
           sData = [sData, '\n', sprintf('%13.2f', nObjData(iDataRow, :))];
70   end;

```

```

if bVerbose,
    if iCurrImg == 1, printf(sCols);, end;
    printf(sData);
75 end;

% output numeric segmentation data to file
sFileName = sprintf('%s%s%03d%s%04d.dat', sObjDataPath, filesep, ...
                    iLocID, filesep, iCurrIdx);
80 if ~exist(sFileName) | bOverwrite,
    if ~exist('fid'), fid = fopen(sFileName, 'w+');, end;
    fprintf(fid, sCols);
    fprintf(fid, sData);
    fclose(fid);
85 clear fid;
end;

if bBlobOut,
    % all analysis types degenerate from single cell quantification.
90 % collect numeric data for blob analysis by averaging each
    % quantification channel over all valid segmented objects
    if iCurrImg == 1, datablb = [];, end;

    % locid, imgid, area, (mean, std, min ,max, bglvl) ...
95 csColBlb = {'LOCID', 'IMGID', 'AREA'};
    cFL = {};
    nFL = [];
    for i = 1:size(cIqnt, 1),
        cFL = {cFL{:}, ...
100             sprintf('MEAN%d', i), ...
             sprintf('STD%d', i), ...
             sprintf('MIN%d', i), ...
             sprintf('MAX%d', i), ...
             sprintf('BGLVL%d', i) ...
105         };
        nFL = [nFL, ...
              mean(getdatacol(sprintf('MEAN%d', i), nObjData, colsscfl)), ...
              mean(getdatacol(sprintf('STD%d', i), nObjData, colsscfl)), ...
              min(getdatacol(sprintf('MIN%d', i), nObjData, colsscfl)), ...
110             max(getdatacol(sprintf('MAX%d', i), nObjData, colsscfl)), ...
              mean(getdatacol(sprintf('BGLVL%d', i), nObjData, colsscfl)) ...
              ];
    end;
    csColBlb = {csColBlb{:}, cFL{:}};
115 datarow = [iLocID, iCurrIdx, ...
             sum(getdatacol('AREA', nObjData, colsscfl)), ...
             nFL ...
             ];
    datablb = [datablb; datarow];
120

    if bVerbose,
        printf('Blob Summary:');
        printf('%13s', csColBlb{:});
        sDataBlb = ['\n', sprintf('%13.2f', datarow)];
125 printf(sDataBlb);
    end;
end;

```

```

% -----
130 % application specific data collection code
% -----
%
135 % END DATA COLLECTION CODE
% =====

tLoopElaps = [tLoopElaps etime(clock, tLoopInit)];
end;
140
if bBlobOut,
    % write the blob data to the output directory. unnecessary to
    % organize by location id here since all image data is one file
    sFileName = sprintf('%s%s%03d.blob.dat', sObjDataPath, filesep, iLocID);
145 if ~exist(sFileName) | bOverwrite,
        % initialize the blob data file for writing
        fidblb = fopen(sFileName, 'w+');

        % write out the column headers
150 fprintf(fidblb, '%13s', csColBlb{:});

        for iDataRow = 1:size(databl, 1);
            fprintf(fidblb, '\n');
            fprintf(fidblb, '%13.2f', databl(iDataRow, :));
155 end;

        fclose(fidblb);
        clear fidblb;
    end;
160 end;

close(hProg);
clear hProg;

```

```

_____ imsegtrack.m _____
function imsegtrack(runinfo, quantinfo, trackinfo, cvChannels)
% stand alone tracking prep and execution script

% Experiment info:
5 % sExptBase      = runinfo.path;
% sExptPrefix     = runinfo.prefix;
% sExptID         = runinfo.id;
% sExptPath       = sprintf('%s%s%s', sExptBase, filesep, ...
%                                     sExptPrefix, sExptID);
10 sExptPath       = runinfo.path;

iLocID           = runinfo.locid;
viIdx            = runinfo.indices;
15 nTotImgs      = length(viIdx);

sObjDataPath     = [sExptPath, filesep, quantinfo.path];

sTrjDataPath     = [sExptPath, filesep, trackinfo.path];
20

```

```

nMinTrajLength = trackinfo.minlength;
if nMinTrajLength <= 1,
    % Evaluate the trajectory length spec as a percent of longest possible
    % trajectory
25     nMinTrajLength = nTotImgs*trackinfo.minlength;

    % otherwise, consider it to be an absolute value for the trajectory
    % length in points
end;
30
nMaxDisp      = trackinfo.maxdisp;
nMem          = trackinfo.mem;
bVerbose      = trackinfo.verbose;
bOverwrite    = trackinfo.overwrite;
35
% create data output directories if needed
if ~exist(sTrjDataPath, 'dir'),
    mkdir(sTrjDataPath);
end;
40 if ~exist(sprintf('%s/%03d', sTrjDataPath, iLocID), 'dir'),
    mkdir(sprintf('%s/%03d', sTrjDataPath, iLocID));
end;
if ~exist(sprintf('%s/%03d/byobj', sTrjDataPath, iLocID), 'dir'),
    mkdir(sprintf('%s/%03d/byobj', sTrjDataPath, iLocID));
45 end;
if ~exist(sprintf('%s/%03d/byimg', sTrjDataPath, iLocID), 'dir'),
    mkdir(sprintf('%s/%03d/byimg', sTrjDataPath, iLocID));
end;

50 % preparation complete, start the collection process.
% collect the data from data files into a single array for import into the
% tracking algorithm
hProg = waitbar(0, '');
% iProgPos = get(hProg, 'position');
55 % set(hProg, 'position', [1, 25, iProgPos(3:4)]);

iCurrImg = 0;
tLoopElaps = [];
nTrjData = [];
60 for iCurrIdx = viIdx,
    iCurrImg = iCurrImg + 1;
    nProgress = iCurrImg/nTotImgs;

    tLoopInit = clock;
65     updateprogress(hProg, iCurrImg, nTotImgs, tLoopElaps);

    waitbar(nProgress, hProg, 'Reading Image Object Data Set');
    data = importdata(sprintf('%s/%03d/%04d.dat', sObjDataPath, ...
                                iLocID, iCurrIdx));
70
    nTrjData = [nTrjData; data.data];

    tLoopElaps = [tLoopElaps etime(clock, tLoopInit)];
end;
75
cols = data.colheaders;

```

```

waitbar(nProgress, hProg, 'Preparing for Object Tracking');

80 % prepare the data for object tracking
oid = getdatacol('objid', nTrjData, cols);
x   = getdatacol('centx', nTrjData, cols);
y   = getdatacol('centy', nTrjData, cols);
t   = getdatacol('imgid', nTrjData, cols);
85 a   = getdatacol('area', nTrjData, cols);
pmm = getdatacol('perimnative', nTrjData, cols);

FL = [];
csFL = {};
90 for i = 1:length(strmatch('qnt', {cvChannels(:, 6)}, 'exact')),
    fl = getdatacol(['mean' num2str(i)], nTrjData, cols);
    st = getdatacol(['std' num2str(i)], nTrjData, cols);
    bg = getdatacol(['bglvl' num2str(i)], nTrjData, cols);
    % FL = [FL, fl./bg-1, st./bg, bg];
95    FL = [FL, fl, st, bg];
    csFL = {csFL{:}, ['FL' num2str(i)], ...
             ['STD' num2str(i)], ...
             ['BG' num2str(i)]};
end;
100 nPreTrackData = [x, y, FL, a, pmm, oid, t];
colspretrack = {'CENTX', 'CENTY', csFL{:}, 'AREA', 'PERIMNATIVE', ...
                'OLDOBJID', 'IMGID'};
colsposttrack = {colspretrack{:}, 'NEWOBJID'};
105 waitbar(nProgress, hProg, 'Tracking Objects ...');
nPostTrackData = track(nPreTrackData, nMaxDisp, ...
                      'dim', 2, ...
                      'mem', nMem, ...
110                      'quiet', ~bVerbose);
csObjData = splitdata(nPostTrackData, colsposttrack, 'newobjid');

waitbar(nProgress, hProg, 'Finding Valid Trajectories');
trjlen = [];
115 for i=1:length(csObjData),
    trjlen = [trjlen; size(csObjData{i}, 1)];
end;
csValTrj = {csObjData{find(trjlen >= nMinTrajLength)}};
printf('Found %d ''Good'' Trajectories', length(csValTrj));
120 % output valid object trajectory data to files
objids = [];
for iTrj = 1:length(csValTrj),
    nProgress = iTrj/length(csValTrj);
125    waitbar(nProgress, hProg, 'Writing Trajectory Data');
    iObjID = unique(getdatacol('newobjid', csValTrj{iTrj}, colsposttrack));
    objids = [objids; iObjID];

130    sData = [];
    for iDataRow = 1:size(csValTrj{iTrj}, 1),
        sData = [sData, '\n', sprintf('%13.2f', ...
                                     csValTrj{iTrj}(iDataRow, :))];
    end;

```



```

135     % output numeric trajectory data to file
        sFileName = sprintf('%s%s%03d%sbyobj%s%04d.obj.dat', ...
                            sTrjDataPath, filesep, iLocID, ...
                            filesep, filesep, iObjID);
140     if ~exist('fid'), fid = fopen(sFileName, 'w+'); end;
        fprintf(fid, '%13s', colsposttrack{:});
        fprintf(fid, sData);
        fclose(fid);
        clear fid;
145 end;

        % output trajectory data on a per image basis for trajectory validation
        % filter the each image's data to objects that have valid trajectories
        waitbar(nProgress, hProg, 'Generating Image Sequence Data');
150     % ismember(getdatacol('newobjid', nPostTrackData, colsposttrack), objids)

        nValidPostTrackData = nPostTrackData(...
            ismember(...
155             getdatacol('newobjid', nPostTrackData, colsposttrack), objids), :);
        csImgData = splitdata(...
            sortrows(nValidPostTrackData, ...
                strmatch('IMGID', colsposttrack, 'exact')), colsposttrack, 'imgid');

160 for iImg = 1:length(csImgData),
        nProgress = iImg/length(csImgData);

        waitbar(nProgress, hProg, 'Generating Image Sequence Data');
        iImgID = unique(getdatacol('imgid', csImgData{iImg}, colsposttrack));
165     % output segmentation channel? ... probably better to have user pick
        % the overlay to use during validation.
        sData = [];
        for iDataRow = 1:size(csImgData{iImg}, 1),
170             sData = [sData, '\n', sprintf('%13.2f', ...
                                                csImgData{iImg}(iDataRow, :))];
        end;

        % output numeric trajectory data to file
175     sFileName = sprintf('%s%s%03d%sbyimg%s%04d.img.dat', ...
                            sTrjDataPath, filesep, iLocID, filesep, filesep, iImgID);
        if ~exist(sFileName) | bOverwrite,
            if ~exist('fid'), fid = fopen(sFileName, 'w+'); end;
            fprintf(fid, '%13s', colsposttrack{:});
180             fprintf(fid, sData);
            fclose(fid);
            clear fid;
        end;
    end;
185     close(hProg);
        clear hProg;

```

```

_____ line2mask.m _____
function B = line2mask(sz, X, Y)
% converts points that define linesegments into a binary mask where lines

```

```

% are set to ON and the rest of the image is OFF

5 % read in point pairs the demarcate line segments
% calculate the slope and intercept of the segment
% redraw the line on the binary pixel by pixel

B = zeros(sz);
10 for i = 1:length(X)-1,
    x = X(i:i+1);
    y = Y(i:i+1);

    xs = 1; if x(2) < x(1), xs = -1; end;
15    ys = 1; if y(2) < y(1), ys = -1; end;

    if diff(x) == 0,
        % vertical lines
        v = [y(1):ys:y(2)];
20        u = ones(size(v))*x(1);
        sType = 'VERT';

    elseif diff(y) == 0,
        % horizontal lines
25        u = [x(1):xs:x(2)];
        v = ones(size(u))*y(1);
        sType = 'HORZ';

    else,
30        % all other angles
        m = (y(2) - y(1))/(x(2) - x(1));
        b = y(1) - m*x(1);

        if abs(diff(y)) > abs(diff(x)),
35            u = linspace(x(1), x(2), abs(diff(y)));
            v = m*u + b;
        elseif abs(diff(y)) < abs(diff(x)),
            v = linspace(y(1), y(2), abs(diff(x)));
            u = (v - b)/m;
40        else,
            u = [x(1):xs:x(2)];
            v = m*u + b;
        end;
        sType = 'ANGL';
45    end;
    u = round(u);
    v = round(v);
    %printf('%s: (%.2f, %.2f) -> (%.2f, %.2f)', sType, u(1), v(1), u(2), v(2));
50    B(sub2ind(sz, v, u)) = 1;
end;

B = bwmorph(B, 'close');
55 B = bwmorph(B, 'skel', inf);

```

```

_____ mergedata.m _____
function [ndata] = mergedata(cdata, dim)
%MERGEDATA

```

```

% [ndata] = mergedata(cdata, dim)
%
5 % Merges cell array data into a numeric array along dimension dim
ndata = cat(dim, cdata{:});


---


_____ normrange.m _____
function ndata=normrange(data)
ndata = (data - min(data(:)))./(max(data(:)) - min(data(:)));


---


_____ parseopts.m _____
function [value] = parseopts(optname, opts, defaultvalue)
% PARSEOPTS
% [value] = parseopts(optname, opts, defaultvalue)
%
5 % Gets the value for 'optname' from options data provided in 'opts'. If
% 'optname' is not found or 'opts' is empty, returns 'defaultvalue'.

value = defaultvalue;
if ~isempty(opts),
10 iOpt = strmatch(lower(optname), opts(:,1), 'exact');
    if ~isempty(iOpt),
        value = opts{iOpt, 2};
    end;
end;


---


_____ printf.m _____
function printf(sFormatString, varargin)
disp(sprintf(sFormatString, varargin{:}));


---


_____ processing.m _____
function I = processing(I, N, F, varargin)
opts = getopts(varargin);
iOffset = parseopts('offset', opts, 2^15);
iChBin = parseopts('binning', opts, 1);
5 iChShift = parseopts('shift', opts, [0 0]);
iROI = parseopts('roi', opts, []);

I = imsubtract(I, iOffset);
if iChBin > 1, I = imbin(I, iChBin);, end;
10
% subtract off the bias offset ... this should also partially remove the
% background level of the image.
I = imsubtract(I, mean(N(:)));

15 % make the flatfield image the same size as I, just in case the image is
% not the same size as the correction field.
F = imresize(F, mean(size(I)./size(F)));
I = double(I)./F;

20 if find(iChShift), I = circshift(I, iChShift);, end;

if ~isempty(iROI),
    I = imcrop(I, iROI);
end;
25
I = uint16(I);


---



```

```

function [cIall, cIseg, cIqnt] = readingset(iLocID, imgid, sExptPath, ...
                                         cvChannels, cvROIs, N, F)

% READINGSET
% [cIall, cIseg, cIqnt] = readingset(iLocID, imgid, sExptPath, ...
5 %                                     cvChannels, cvROIs, N, F)
%
% Reads in images from an acquisition set comprised of multiple imaging
% channels defined by the cell array cvChannels and stores them in hash
% tables:
10 %
% cIall    Qx3 cell array of all channels read, where Q is the number of
%          channels. Columns are channel name, image data, and channel
%          role - e.g. 'seg' for segmentation.
%
15 % cIseg    Px2 cell array of channels used for segmentation. Columns are
%          channel name and image data.
%
% cIqnt    Nx2 cell array of channels used quantification. Columns are
%          channel name and image data.
20 %
% cIseg and cIqnt can be regenerated from cIall by returning the first
% two columns of a search on the third column for 'seg' and 'qnt',
% respectively.
%
25 % cvChannels is a table of channel information with the following
% structure:
%
%          NAME    NOISE    FLAT    BIN          SHIFT          ROLE    ROI
%          String  String   String <1x1 int> <1x2 int>   String  String
30 %
% The ROI column of cvChannels contains a string that corresponds to a
% string key in the cvROI table:
%
%          NAME    ROI Spec
35 %          String <1x4 double>
%
% ROI Spec is the same as RECT used/output by IMCROP.
%
% See Also:
40 % processing(), createcorrimg(), imbin(), circshift(), imcrop()

% HISTORY
%      2006/02/23, WLP:    Created.
%      2006/03/13, WLP:    Added ROI cell array input for better ROI
45 %                        selection and management.
%      2006/10/01, WLP:    Channel names that are 'none' or blank are
%                        skipped. Before they would crash the program.
%

50 cIall = {}; cIseg = {}; cIqnt = {};
   csCorrNoise = N(:, 1);
   csCorrFlat = F(:, 1);
   csROIs = cvROIs(:, 1);
   for j = 1:size(cvChannels, 1),
55     cIall{j, 1} = cvChannels{j, 1};
       cIall{j, 2} = [];

```

```

cIall{j, 3} = cvChannels{j, 6};
if exist([sExptPath '/' cvChannels{j, 1}], 'dir') ...
    && (~strcmpi(cvChannels{j, 1}, 'none') ...
60    || ~isempty(strtrim(cvChannels{j, 1}))),
    cIall{j, 2} = getimg(cvChannels{j, 1}, iLocID, imgid, ...
                        'basepath', sExptPath);
    In          = N{strmatch(cvChannels{j, 2}, csCorrNoise, 'exact'), 2};
    If          = F{strmatch(cvChannels{j, 3}, csCorrFlat, 'exact'), 2};
65    bin        = cvChannels{j, 4};
    shft       = cvChannels{j, 5};
    roi        = cvROIs{strmatch(cvChannels{j, 7}, csROIs, 'exact'), 2};

    cIall{j, 2} = processing(cIall{j, 2}, In, If, ...
70    'binning', bin, 'shift', shft, 'roi', roi);
end;
end;
cIseg = cIall(strmatch('seg', cIall(:, 3), 'exact'), 1:2);
cIqnt = cIall(strmatch('qnt', cIall(:, 3), 'exact'), 1:2);

```

```

_____ splitdata.m _____
function [cdata]=splitdata(data, columns, sSplitColumn)
%SPLITDATA
% [cdata]=splitdata(data, columns, sSplitColumn)
%
5 % splits a data list by unique values in a specified column. split data
% is returned as a cell array.

iObjList = data(:,strmatch(upper(sSplitColumn), upper(columns)));
iObjID = unique(iObjList);
10 cdata = {};
for i=1:length(iObjID),
    iFirstIdx = find(iObjList == iObjID(i), 1, 'first');
    iLastIdx = find(iObjList == iObjID(i), 1, 'last');

15 cdata{i} = data(iFirstIdx:iLastIdx, :);
end;

```

```

_____ track.m _____
function tracks = track(xyzs,maxdisp,varargin)
% TRACK
% result = track(positionlist, maxdisp)
% result = track(positionlist, maxdisp, ...)
5 %
% Constructs n-dimensional trajectories from a scrambled list of
% particle coordinates determined at discrete times (e.g. in
% consecutive video frames).
%
10 % see http://glinda.lrsm.upenn.edu/~weeks/idl for more information
%
% CATEGORY:
% Image Processing
%
15 % INPUTS:
% positionlist
% an array listing the scrambled coordinates and data of the
% different particles at different times, such that:
%

```

```

20 %           positionlist(:,1:d)
%               contains the d coordinates and data for all the particles,
%               at the different times. must be positive
%           positionlist(:,end)
%               contains the time t that the position was determined, must
25 %               be integers (e.g. frame number. These values must be
%               monotonically increasing and uniformly gridded in time.
%
%           simply put, the data array must have positional coordinates in the
%           first columns and a time vector in the last column.
30 %
% maxdisp
%     an estimate of the maximum distance that a particle would move in a
%     single time interval.(see Restrictions)
%
35 % OPTIONAL INPUTS:
% Optional inputs are passed to the function as (parameter, value) pairs -
% e.g.:
%
%           result = track(posdata, maxdisp, 'mem', 5, 'dim', 3)
40 %
% If an optional argument is not specified then it's default value is
% used. In the example above, the 'mem' parameter is set to 5 and the
% 'dim' parameter is set to 3. The other options - 'goodenough' and
% 'quiet' - will have default values of 0, and false.
45 %
% mem           integer default = 0
%               this is the number of time steps that a particle can be 'lost' and
%               then recovered again. If the particle reappears after this number
%               of frames has elapsed, it will be tracked as a new particle. This
50 %               is useful if particles occasionally 'drop out' of the data.
%
% dim           integer default = 2
%               if the user would like to unscramble non-coordinate data for the
%               particles (e.g. apparent radius of gyration for the particle
55 %               images), then positionlist should contain the position data in
%               positionlist(0:param.dim-1,*) and the extra data in
%               positionlist(param.dim:d-1,*). It is then necessary to set dim
%               equal to the dimensionality of the coordinate data to so that the
%               track knows to ignore the non-coordinate data in the construction
60 %               of the trajectories.
%
% goodenough    integer default = 0
%               set this keyword to eliminate all trajectories with fewer than
%               'goodenough' valid positions. This is useful for eliminating very
65 %               short, mostly 'lost' trajectories due to blinking 'noise' particles
%               in the data stream.
%
% quiet         logical default = false
%               set this 'true' if you don't want any text
70 %
%
% OUTPUTS:
% result
%           A list containing the original data rows sorted into a series of
75 %           trajectories. An additional column containing a unique id number
%           for each identified particle trajectory is appended to the original

```

```

%      input data.  The result array is sorted so rows with corresponding
%      id numbers are in contiguous blocks sorted by increasing time.
%
80 %      For example:
%
%      For the input data structure (positionlist):
%      (x)      (y)      (t)
%      pos = 3.60000      5.00000      0.00000
85 %      15.1000      22.6000      0.00000
%      4.10000      5.50000      1.00000
%      15.9000      20.7000      2.00000
%      6.20000      4.30000      2.00000
%
90 %      >> res = track(pos, 5, 'mem', 2)
%
%      track will return the result 'res'
%      (x)      (y)      (t)      (id)
%      res = 3.60000      5.00000      0.00000      0.00000
95 %      4.10000      5.50000      1.00000      0.00000
%      6.20000      4.30000      2.00000      0.00000
%      15.1000      22.6000      0.00000      1.00000
%      15.9000      20.7000      2.00000      1.00000
%
100 %      for t=1 in the example above, one particle temporarily vanished.  As
%      a result, the trajectory id=1 has one time missing, i.e. particle
%      loss can cause time gaps to occur in the corresponding trajectory
%      list.  In contrast:
%
105 %      >> res = track(pos,5)
%
%      track will return the result 'res'
%      (x)      (y)      (t)      (id)
%      res = 15.1000      22.6000      0.00000      0.00000
110 %      3.60000      5.00000      0.00000      1.00000
%      4.10000      5.50000      1.00000      1.00000
%      6.20000      4.30000      2.00000      1.00000
%      15.9000      20.7000      2.00000      2.00000
%
115 %      where the reappeared 'particle' will be labelled as new rather than
%      as a continuation of an old particle since mem=0.  It is up to the
%      user to decide what setting of 'mem' will yeild the highest
%      fidelity.
%
120 %      SIDE EFFECTS:
%      Produces informational messages.  Can be memory intensive for
%      extremely large data sets.
%
%      RESTRICTIONS:
125 %      maxdisp should be set to a value somewhat less than the mean spacing
%      between the particles.  As maxdisp approaches the mean spacing the
%      runtime will increase significantly.  The function will produce an
%      error message: "Excessive Combinatorics!" if the run time would be
%      too long, and the user should respond by re-executing the function
130 %      with a smaller value of maxdisp.  Obviously, if the particles being
%      tracked are frequently moving as much as their mean separation in a
%      single time step, this function will not return acceptable
%      trajectories.

```

```

%
135 % PROCEDURE:
%       Given the positions for n particles at time t(i), and m possible
%       new positions at time t(i+1), this function considers all possible
%       identifications of the n old positions with the m new positions,
%       and chooses that identification which results in the minimal total
140 % squared displacement. Those identifications which don't associate a
%       new position within maxdisp of an old position (particle loss)
%       penalize the total squared displacement by maxdisp^2. For non-
%       interacting Brownian particles with the same diffusivity, this
%       algorithm will produce the most probable set of identifications
145 % (provided maxdisp >> RMS displacement between frames).
%
%       In practice it works reasonably well for systems with oscillatory,
%       ballistic, correlated and random hopping motion, so long as single
%       time step displacements are reasonably small.
150 %
%       Multidimensional functionality is intended to facilitate tracking
%       when additional information regarding target identity is available
%       (e.g. size or color). At present, this information should be
%       rescaled by the user to have a comparable or smaller (measurement)
155 % variance than the spatial displacements.
%
% MODIFICATION HISTORY:
%   2/93 Written by John C. Crocker, University of Chicago (JFI).
%   7/93 JCC fixed bug causing particle loss and improved performance
160 %       for large numbers of (>100) particles.
%   11/93 JCC improved speed and memory performance for large
%       numbers of (>1000) particles (added subnetwork code).
%   3/94 JCC optimized run time for trivial bonds and d<7. (Added
%       d-dimensional raster metric code.)
165 %   8/94 JCC added functionality to unscramble non-position data
%       along with position data.
%   9/94 JCC rewrote subnetwork code and wrote new, more efficient
%       permutation code.
%   5/95 JCC debugged subnetwork and excessive combinatorics code.
170 %   12/95 JCC added memory keyword, and enabled the tracking of
%       newly appeared particles.
%   3/96 JCC made inipos a keyword, and disabled the adding of 'new'
%       particles when inipos was set.
%   3/97 JCC added 'add' keyword, since Chicago users didn't like
175 %       having particle addition be the default.
%   9/97 JCC added 'goodenough' keyword to improve memory efficiency
%       when using the 'add' keyword and to filter out bad tracks.
%   10/97 JCC streamlined data structure to speed runtime for >200
%       timesteps. Changed 'quiet' keyword to 'verbose'. Made
180 %       time labelling more flexible (uniform and sorted is ok).
%   9/98 JCC switched trajectory data structure to a 'list' form,
%       resolving memory issue for large, noisy datasets.
%   2/99 JCC added Eric Weeks's 'uberize' code to post-facto
%       rationalize the particle id numbers, removed 'add' keyword.
185 %
%   1/05 Transmuted to MATLAB by D. Blair
%   5/05 ERD Added the param structure to simplify calling.
%   6/05 ERD Added quiet to param structure
%   7/05 DLB Fixed slight bug in trivial bond code
190 %

```



```

% 11/05 Studied by W. L. Pang University of California, San Diego
% 11/05 WLP Changed the way optional parameters are specified. It now
%         uses the 'varargin' cell array and (param,value) pairs instead
%         of a param structure.
195 % 11/05 WLP Cleaned up the header comments to be more MATLAB-like and more
%         readable.
% 12/01 WLP Changed the validation of the time vector. If it is not in
%         order the algorithm sorts the positionlist data by row so that
%         the time vector is in monotonically increasing order.
200 % 12/01 WLP Replaced directly implanted UNQ code with the built-in Matlab
%         function 'unique()'.
%
% This code 'track.pro' is copyright 1999, by John C. Crocker.
% It should be considered 'freeware'- and may be distributed freely
205 % (outside of the military-industrial complex) in its original form
% when properly attributed.

% dd = length(xyzs(1,:));
dd = size(xyzs, 2);
210
% get user options if there are any
% use default parameters if none given
opts      = getopts(varargin);

215 % if mem is not needed set to zero
memory_b  = parseopts('mem', opts, 0);

% if goodenough is not wanted set to zero
goodenough = parseopts('goodenough', opts, 0);
220
dim        = parseopts('dim', opts, dd - 1);
quiet      = parseopts('quiet', opts, false);
inipos     = parseopts('inipos', opts, []);

225 % % checking the input time vector
% t = xyzs(:,dd);
% st = circshift(t,1);
% st = t(2:end) - st(2:end);
% if sum(st(find(st < 0))) ~= 0
230 %     disp('The time vector is not in order')
%
%     return
% end
% info = 1;
235
% WLP
% why not sort the array if the time vector is not in order rather than
% just giving up? The algorithm already assumes a randomly scrambled
% set of positions. Using native matlab functions would be easier:
240 if ~issorted(xyzs(:, end)),
    xyzs = sortrows(xyzs, dd);
end;
t = xyzs(:, end);
st = circshift(t, 1);
245 st = t(2:end) - st(2:end);

w = find(st > 0);

```

```

z = length(w) + 1;
if isempty(w)
250   disp('All positions are at the same time... go back!')
      return
end

% partitioning the data with unique times
255 % %res = unq(t);
% % implanting unq directly
%   indices = find(t ~= circshift(t,-1));
%   count = length(indices);
%   if count > 0
260 %       res = indices;
%   else
%       res = length(t) -1;
%   end
% %%%%%%%%%%%%%%%
265 [uniquevals, res] = unique(t); clear uniquevals;
res = [1,res',length(t)];

% get initial positions
ngood = res(2) - res(1) + 1;
270 eyes = 1:ngood;

% if keyword_set( inipos ) then begin
%   pos = inipos(0:dim-1,*);
%   istart = 0L
275 %   n = n_elements(pos(0,*))
% endif else begin
%   pos = xyzs(0:dim-1,eyes)
%   istart = 1L      ;we don't need to track t=0.
%   n = ngood
280 % endelse

if ~isempty(inipos),
    pos = inipos(:, 1:dim);
    istart = 1;
285   n = size(pos, 1);
else,
    pos = xyzs(eyes,1:dim);
    istart = 2;      % we don't need to track t=0
    n = ngood;
290 end;

% how long are the 'working' copies of the data?
zspan = 50;
if n > 200
295   zspan = 20;
end
if n > 500
    zspan = 10;
end
300 resx = zeros(zspan,n) - 1;

bigresx = zeros(z,n) - 1;
mem = zeros(n,1);
% whos resx

```

```

305 % whos bigresx
    uniqid = 1:n;
    maxid = n;
    olist = [0.,0.];

310 if goodenough > 0
    dumphash = zeros(n,1);
    nvalid = ones(n,1);
    end

315 % whos eyes;
    resx(1,:) = eyes;
    % setting up constants
    maxdisq = maxdisp^2;

320 % John calls this the setup for "fancy code" ???
    notnsqrd = (sqrt(n*ngood) > 200) & (dim < 7);
    notnsqrd = notnsqrd(1);

    if notnsqrd
325     %;    construct the vertices of a 3x3x3... d-dimensional hypercube

        cube = zeros(3^dim,dim);

330     for d=0:dim-1,
        numb = 0;
        for j=0:(3^d):(3^dim)-1,
            cube(j+1:j+(3^d),d+1) = numb;
            numb = mod(numb+1,3);
335     end
        end

        % calculate a blocksize which may be greater than maxdisp, but which
        % keeps nblocks reasonably small.

340     volume = 1;
        for d = 0:dim-1
            minn = min(xyzs(w,d+1));
            maxx = max(xyzs(w,d+1));
345     volume = volume * (maxx-minn);
        end
        volume;
        blocksize = max( [maxdisp,((volume)/(20*ngood))^(1.0/dim)] );
    end

350 % Start the main loop over the frames.
    for i=istart:z
        ispan = mod(i-1,zspan)+1;
        %disp(ispan)
        % get new particle positions
355     m = res(i+1) - res(i);
        res(i);
        eyes = 1:m;
        eyes = eyes + res(i);

360     if m > 0

```

```

xyi = xyzs(eyes,1:dim);
found = zeros(m,1);

365 % THE TRIVIAL BOND CODE BEGINS
%   if ~quiet, printf('Entering trivial bond code ...'); end;

if notnsqrd
    %Use the raster metric code to do trivial bonds
370
    % construct "s", a one dimensional parameterization of the space
    % which consists of the d-dimensional raster scan of the volume.)

    abi = fix(xyi./blocksize);
    abpos = fix(pos./blocksize);
375    si = zeros(m,1);
    spos = zeros(n,1);
    dimm = zeros(dim,1);
    coff = 1.;

380    for j=1:dim
        minn = min([abi(:,j);abpos(:,j)]);
        maxx = max([abi(:,j);abpos(:,j)]);
        abi(:,j) = abi(:,j) - minn;
385        abpos(:,j) = abpos(:,j) - minn;
        dimm(j) = maxx-minn + 1;
        si = si + abi(:,j).*coff;
        spos = spos + abpos(:,j).*coff;
        coff = dimm(j).*coff;

390    end
    nblocks = coff;
    % trim down (intersect) the hypercube if its too big to fit in
    % the particle volume. (i.e. if dimm(j) lt 3)

395    cub = cube;
    deg = find( dimm < 3);
    if ~isempty(deg)
        for j = 0:length(deg)-1
            cub = cub(find(cub(:,deg(j+1)) < dimm(deg(j+1))));
400        end
    end

    % calculate the "s" coordinates of hypercube (with a corner @
    % the origin)
405    scube = zeros(length(cub(:,1)),1);
    coff = 1;
    for j=1:dim
        scube = scube + cub(:,j).*coff;
        coff = coff*dimm(j);

410    end

    % shift the hypercube "s" coordinates to be centered around the
    % origin

415    coff = 1;
    for j=1:dim
        if dimm(j) > 3
            scube = scube - coff;

```

```

        end
420     coff = dimm(j).* coff;
    end
    scube = mod((scube + nblocks),nblocks);
    % get the sorting for the particles by their "s" positions.
    [ed,isort] = sort(si);
425
    % make a hash table which will allow us to know which new
    % particles are at a given si.
    strt = zeros(nblocks,1) -1;
    fnsh = zeros(nblocks,1);
430    h = find(si == 0);
    lh = length(h);
    if lh > 0

435    si(h) = 1;
    end

    for j=1:m
        if strt(si(isort(j))) == -1
            strt(si(isort(j))) = j;
            fnsh(si(isort(j))) = j;
440        else
            fnsh(si(isort(j))) = j;
        end
    end
    end
445    if lh > 0
    si(h) = 0;
    end
    coltot = zeros(m,1);
    rowtot = zeros(n,1);
450    which1 = zeros(n,1);
    for j=1:n

        map = fix(-1);
455

        scub_spos = scube + spos(j);
        s = mod(scub_spos,nblocks);
        whzero = find(s == 0 );
        if ~isempty(whzero)
460            nfk = find(s ~=0);
            s = s(nfk);
        end

        w = find(strt(s) ~= -1);
465

        ngood = length(w);
        ltmax=0;
        if ngood ~= 0

470            s = s(w);
            for k=1:ngood
                map = [map;isort( strt(s(k)):fnsh(s(k)))]];
            end
            map = map(2:end);
475 %         if length(map) == 2

```

```

%             if (map(1) - map(2)) == 0
%                 map = unique(map);
%             end
%         end
480     % map = map(umap);
%end
% find those trival bonds
distq = zeros(length(map),1);
for d=1:dim
485     distq = distq + (xyi(map,d) - pos(j,d)).^2;
end
ltmax = distq < maxdisq;

rowtot(j) = sum(ltmax);

490     if rowtot(j) >= 1
        w = find(ltmax == 1);
        coltot( map(w) ) = coltot( map(w) ) +1;
        which1(j) = map( w(1) );
495     end
end

end

500

ntrk = fix(n - sum(rowtot == 0));

w = find( rowtot == 1);
ngood = length(w);

505

if ngood ~= 0
    ww = find(coltot( which1(w) ) == 1);
    ngood = length(ww);
510    if ngood ~= 0
        %disp(size(w(ww)))
        resx(ispan,w(ww)) = eyes( which1(w(ww)));
        found( which1( w(ww) ) ) = 1;
        rowtot( w(ww) ) = 0;
515        coltot( which1(w(ww)) ) = 0;
    end
end

labely = find( rowtot > 0);
520 ngood = length(labely);
if ngood ~= 0
    labelx = find( coltot > 0);

    nontrivial = 1;
525 else
    nontrivial = 0;
end

else

530

% or: Use simple N^2 time routine to calculate trival bonds
% if ~quiet, printf('Using simple N^2 routine ...'); end;

```

```

% let's try a nice, loopless way!
% don't bother tracking perm. lost guys.
535 wh = find( pos(:,1) >= 0);
ntrack = length(wh);
if ntrack == 0
    printf('WARNING: No valid particles to track!');
    break
540 end
xmat = zeros(ntrack,m);
count = 0;
for kk=1:ntrack
    for ll=1:m
545         xmat(kk,ll) = count;
        count = count+1;
    end
end
count = 0;
550 for kk=1:m
    for ll=1:ntrack
        ymat(kk,ll) = count;
        count = count+1;
    end
555 end

xmat = (mod(xmat,m) + 1);
ymat = (mod(ymat,ntrack) + 1)';
[lenxn,lenxm] = size(xmat);
560 % whos ymat
% whos xmat
% disp(m)

for d=1:dim
565     x = xyi(:,d);
        y = pos(wh,d);
        xm = x(xmat);
        ym = y(ymat(1:lenxn,1:lenxm));
        if size(xm) ~= size(ym)
570             xm = xm';
        end

        if d == 1
            dq = (xm -ym).^2;
575             %dq = (x(xmat)-y(ymat(1:lenxn,1:lenxm))).^2;
        else
            dq = dq + (xm-ym).^2;
            %dq = dq + (x(xmat)-y(ymat(1:lenxn,1:lenxm)) ).^2;
        end
580     end

ltmax = dq < maxdisq;

% figure out which trivial bonds go with which
585 rowtot = zeros(n,1);
rowtot(wh) = sum(ltmax,2);

```

```

590         if ntrack > 1
                coltot = sum(ltmax,1);
            else
                coltot = ltmax;
            end
595         which1 = zeros(n,1);
            for j=1:ntrack
                [mx, w] = max(ltmax(j,:));
                which1(which1(j)) = w;
            end
600
            ntrk = fix( n - sum(rowtot == 0));
            w= find( rowtot == 1 ) ;
            ngood = length(w);
            if ngood ~= 0
605                 ww = find(coltot(which1(w)) == 1);
                    ngood = length(ww);
                    if ngood ~= 0
                        resx( ispan, w(ww) ) = eyes( which1( w(ww)));
                        found(which1( w(ww))) = 1;
610                         rowtot(w(ww)) = 0;
                        coltot(which1(w(ww))) = 0;
                    end
                end
            end

615         labely = find( rowtot > 0);
            ngood = length(labely);

            if ngood ~= 0
                labelx = find( coltot > 0);
620                 nontrivial = 1;
            else
                nontrivial = 0;
            end
        end
end
625
%THE TRIVIAL BOND CODE ENDS

if nontrivial
    %
630     if ~quiet, printf('Entering non-trivial bond code ...'); end;

        xdim = length(labelx);
        ydim = length(labely);

        % make a list of the non-trivial bonds
635
        bonds = zeros(1,2);
        bondlen = 0;

        for j=1:ydim
640             distq = zeros(xdim,1);

                for d=1:xdim
                    %distq
                    distq = distq + (xyi(labelx,d) - pos(labely(j),d)).^2;
645                 %distq
                end
            end
        end
    end
end

```



```

w= find(distq < maxdisq)' - 1;
ngood = length(w);
650 newb = [w;(zeros(1,ngood)+j)];

bonds = [bonds;newb'];

655 bondlen = [ bondlen;distq( w + 1) ];

end
bonds = bonds(2:end,:);

660 bondlen = bondlen(2:end);
numbonds = length(bonds(:,1));
mbonds = bonds;
max([xdim,ydim]);

665

if max([xdim,ydim]) < 4
    nclust = 1;
    maxsz = 0;
    mxsz = xdim;
670    mysz = ydim;
    bmap = zeros(length(bonds(:,1))+1,1) - 1;

else

675
    % THE SUBNETWORK CODE BEGINS
    % if ~quiet, printf('Entering subnetwork code ...'); end;
    lista = zeros(numbonds,1);
    listb = zeros(numbonds,1);
680    nclust = 0;
    maxsz = 0;
    thru = xdim;

    while thru ~= 0
685        % the following code extracts connected
        % sub-networks of the non-trivial
        % bonds. NB: lista/b can have redundant entries due
        % to multiple-connected subnetworks

690
        w = find(bonds(:,2) >= 0);
        % size(w)

        lista(1) = bonds(w(1),2);
695        listb(1) = bonds(w(1),1);
        bonds(w(1),:) = -(nclust+1);
        bonds;
        adda = 1;
        addb = 1;
700        donea = 0;
        doneb = 0;
        if (donea ~= adda) | (doneb ~= addb)
            true = 0;

```

```

else
705 true = 1;
end

while ~true

710     if (donea ~= adda)
        w = find(bonds(:,2) == lista(donea+1));
        ngood = length(w);
        if ngood ~= 0
            listb(addb+1:addb+ngood,1) = bonds(w,1);
715         bonds(w,:) = -(nclust+1);
            addb = addb+ngood;
        end
        donea = donea+1;
    end
    if (doneb ~= addb)
720         w = find(bonds(:,1) == listb(doneb+1));
        ngood = length(w);
        if ngood ~= 0
            lista(adda+1:adda+ngood,1) = bonds(w,2);
725         bonds(w,:) = -(nclust+1);
            adda = adda+ngood;
        end
        doneb = doneb+1;
    end
    if (donea ~= adda) | (doneb ~= addb)
730         true = 0;
    else
        true = 1;
    end
735 end

[pp,pqx] = sort(listb(1:doneb));
%unx = unq(listb(1:doneb),pqx);
%b
740     arr = listb(1:doneb);
        q = arr(pqx);
        indices = find(q ~= circshift(q,-1));
        count = length(indices);
        if count > 0
745             unx = pqx(indices);
        else
            unx = length(q) -1;
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
750     xsz = length(unx);
        % xsz = n_elements( unq( listb( 0:doneb-1 ), sort( listb( 0:doneb-1 ) ) ) )
        %

755     [pp,pqy] = sort(lista(1:donea));
        %uny = unq(lista(1:donea),pqy);
        %implanting unq directly
        arr = lista(1:donea);
        q = arr(pqy);
760         indices = find(q ~= circshift(q,-1));

```



```

nold = length(uold);
820
%un = unq(bonds(:,2));

%implanting unq directly
indices = find(bonds(:,2) ~= circshift(bonds(:,2),-1));
825
count = length(indices);
    if count > 0
        un = indices;
    else
        un = length(bonds(:,2)) -1;
830
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

unew = bonds(un,2);
nnew = length(unew);

835
if nnew > 5
    rnsteps = 1;
    for ii =1:nnew
        rnsteps = rnsteps * length( find(bonds(:,2) == ...
840
            unew(ii)));
        if rnsteps > 5.e+4
            disp(['Warning: '...
                'difficult combinatorics encountered.'])
        end
        if rnsteps > 2.e+5
845
            disp(['Excessive Combinitorics you '...
                'FOOL LOOK WHAT YOU HAVE' ...
                ' DONE TO ME!!!'])
            return
        end
850
    end
end
end
st = zeros(nnew,1);
fi = zeros(nnew,1);
855
h = zeros(nbonds,1);
ok = ones(nold,1);
nlost = (nnew - nold) > 0;

for ii=1:nold
860
    h(find(bonds(:,1) == uold(ii))) = ii;
end
st(1) = 1 ;
fi(nnew) = nbonds; % check this later
865
if nnew > 1
    sb = bonds(:,2);
    sbr = circshift(sb,1);
    sbl = circshift(sb,-1);
    st(2:end) = find( sb(2:end) ~= sbr(2:end)) + 1;
870
    fi(1:nnew-1) = find( sb(1:nbonds-1) ~= sbl(1:nbonds-1));
end
%
%
%
    if i-1 == 13
        hi
    end

```



```

                                ok(h(pt(who+1))) = 1;
                                end
                                pt(who+1) = st(who+1) -1;
935                                if lost(who+1)
                                        lost(who+1) = 0;
                                        losttot = losttot -1;
                                end
                                who = who -1;
940                                end
                                end
                                else
                                if ~lost(who+1) & (losttot ~= nlost)
                                lost(who+1) = 1;
945                                losttot = losttot + 1;
                                if pt(who+1) ~= st(who+1)-1
                                        ok(h(pt(who+1))) = 1;
                                end
                                if who == nnew -1
950                                ww = find( lost == 0);
                                dsq = sum(lensq(pt(ww))) ...
                                        + losttot*maxdisq;

                                if dsq < mndisq
955                                minbonds = pt(ww);
                                mndisq = dsq;
                                end
                                else
                                who = who + 1;
960                                end
                                else
                                if pt(who+1) ~= st(who+1) -1
                                        ok(h(pt(who+1))) = 1;
                                end
965                                pt(who+1) = st(who+1) -1;
                                if lost(who+1)
                                        lost(who+1) = 0;
                                        losttot = losttot -1;
                                end
970                                who = who -1;
                                end
                                end
                                end
                                end

975                                checkflag = checkflag + 1;
                                if checkflag == 1
                                        plost = min([fix(mndisq/maxdisq) , (nnew -1)]);
                                        if plost > nlost
                                                nlost = plost;
980                                else
                                        checkflag = 2;
                                end
                                end
                                end

985                                end
                                % update resx using the minimum bond configuration
                                resx(ispan,labely(bonds(minbonds,2))) = ...

```

```

                                eyes(labelx(bonds(minbonds,1)+1));
990         found(labelx(bonds(minbonds,1)+1)) = 1;

        end

        %   THE PERMUTATION CODE ENDS
995     end

    sMsgNetType = 'only trivial networks';

    w = find(resx(ispan,:) >= 0);
1000    nww = length(w);

    if nww > 0
        pos(w,:) = xyzs( resx(ispan,w) , 1:dim);
        if goodenough > 0
1005            nvalid(w) = nvalid(w) + 1;
        end
    end %go back and add goodenough keyword thing
    newguys = find(found == 0);

1010    nnew = length(newguys);

    if (nnew > 0) % & another keyword to workout inipos
        newarr = zeros(zspan,nnew) -1;
        resx = [resx,newarr];

1015        resx(ispan,n+1:end) = eyes(newguys);
        pos = [[pos];[xyzs(eyes(newguys),1:dim)]];
        nmem = zeros(nnew,1);
        mem = [mem;nmem];
1020        nun = 1:nnew;
        unqid = [unqid,((nun) + maxid)];
        maxid = maxid + nnew;
        if goodenough > 0
            dumphash = [dumphash;zeros(1,nnew)'];
1025            nvalid = [nvalid;zeros(1,nnew)'+1];
        end
        % put in goodenough
        n = n + nnew;

1030    end

    else
        ' Warning- No positions found for t='
    end
1035    w = find( resx(ispan,:) ~= -1);
    nok = length(w);
    if nok ~= 0
        mem(w) = 0;
    end

1040    mem = mem + (resx(ispan,:) == -1);
    wlost = find(mem == memory_b+1);
    nlost =length(wlost);

1045    if nlost > 0

```

```

pos(wlost,:) = -maxdisp;
if goodenough > 0
    wdump = find(nvalid(wlost) < goodenough);
    ndump = length(wdump);
1050     if ndump > 0
            dumphash(wlost(wdump)) = 1;
        end
    end
    % put in goodenough keyword stuff if
1055 end
if (ispan == zspan) | (i == z)
    nold = length(bigresx(1,:));
    nnew = n-nold;
    if nnew > 0
1060         newarr = zeros(z,nnew) -1;
            bigresx = [bigresx,newarr];
        end
    if goodenough > 0
        if (sum(dumphash)) > 0
1065             wkeep = find(dumphash == 0);
                nkeep = length(wkeep);
                resx = resx(:,wkeep);
                bigresx = bigresx(:,wkeep);
                pos = pos(wkeep,:);
1070             mem = mem(wkeep);
                uniqid = uniqid(wkeep);
                nvalid = nvalid(wkeep);
                n = nkeep;
                dumphash = zeros(nkeep,1);
1075         end
        end
    end

    % again goodenough keyword
    if quiet~=1
1080         if ntrk == 1, sPPlural = ','; else sPPlural = 's'; end;
            if n == 1, sTPlural = ''; else sTPlural = 's'; end;
            printf('%4d of %d:\t%4d particle%s %4d track%s. %s',...
                i, z, ntrk, sPPlural, n, sTPlural, sMsgNetType...
                );
1085     end
    bigresx(i-(ispan)+1:i,:) = resx(1:ispan,:);
    resx = zeros(zspan,n) - 1;

1090     wpull = find(pos(:,1) == -maxdisp);
        npull = length(wpull);

    if npull > 0
        lillist = zeros(1,2);
1095         for ipull=1:npull
            wpull2 = find(bigresx(:,wpull(ipull)) ~= -1);
                npull2 = length(wpull2);
                thing = [bigresx(wpull2,wpull(ipull)),...
                    zeros(npull2,1)+uniqid(wpull(ipull))];
1100             lillist = [lillist;thing];
        end
    end

```



```

        olist = [[olist];[lillist(2:end,:)]];
1105     end

        wkeep = find(pos(:,1) >= 0);
1110     nkeep = length(wkeep);
        if nkeep == 0
            disp('Were going to crash now, no particles....');
        end
        resx = resx(:,wkeep);
1115     bigresx = bigresx(:,wkeep);
        pos = pos(wkeep,:);
        mem = mem(wkeep);
        uniqid = uniqid(wkeep);
        n = nkeep;
1120     dumphash = zeros(nkeep,1);
        if goodenough > 0
            nvalid = nvalid(wkeep);
        end
    end
1125     end

    if goodenough > 0
        nvalid = sum(bigresx >= 0 ,1);
1130     wkeep = find(nvalid >= goodenough);
        nkeep = length(wkeep);
        if nkeep < n
            bigresx = bigresx(:,wkeep);
            n = nkeep;
1135     uniqid = uniqid(wkeep);
            pos = pos(wkeep,:);
        end
    end
1140     end

    wpull = find( pos(:,1) ~= -2*maxdisp);
    npull = length(wpull);
    if npull > 0
        lillist = zeros(1,2);
1145     for ipull=1:npull
            wpull2 = find(bigresx(:,wpull(ipull)) ~= -1);
            npull2 = length(wpull2);
            thing = [bigresx(wpull2,wpull(ipull)),...
                    zeros(npull2,1)+uniqid(wpull(ipull))];
1150     lillist = [lillist;thing];
        end
        olist = [olist;lillist(2:end,:)];
    end

1155 olist = olist(2:end,:);
    %bigresx = 0;
    %resx = 0;

    nolist = length(olist(:,1));

```

```

1160 res = zeros(nolist,dd+1);
    for j=1:dd
        res(:,j) = xyzs(olist(:,1),j);
    end
    res(:,dd+1) = olist(:,2);
1165
    % this is uberize included for simplicity of a single monolithic code

    ndat=length(res(1,:));
    newtracks=res;
1170

    %u=unq(newtracks(:,ndat));

    % inserting unq
1175 indices = find(newtracks(:,ndat) ~= circshift(newtracks(:,ndat),-1));
        count = length(indices);
        if count > 0
            u = indices;
        else
1180             u = length(newtracks(:,ndat)) -1;
        end

        ntracks=length(u);
1185 u=[0;u];
        for i=2:ntracks+1
            newtracks(u(i-1)+1:u(i),ndat) = i-1;
        end

1190 % end of uberize code

    tracks = newtracks;

```

```

_____ tracklink.m _____
function [dataout]=tracklink(locid, objid, varargin)
% TRACKLINK
% [dataout] = tracklink(locid, objid, ...)
%
5 % Links object trajectory data using a link table. Trajectories start with
% object <objid>. The link table format is:
%
% IMGID          OBJID
% 0              <objid>
10 % :            :
% <ending_imgid> <ending_objid>
%
% Linking behavior depends on the value of OBJID:
%
15 % <int>        The object was reassigned a new objid <int> at <imgid>.
%
% -1           Tracking lost the object at <imgid>. The next row in the link
%              table specifies when the object returns and the object's new
%              objid. Data between the <imgid> values is skipped.
20 %
% #           The object was completely lost at <imgid>. A value of # is
%              specified with the last imgid of the sequence if the object is

```

```

%           retained until the end of the sequence.

25 % DOCUMENT HISTORY
% 2005/??/??   WLP       Created.
% 2006/09/06   WLP       Changed linking behavior to include a trajectory
%                       terminator character (#, which is read as NaN).

30 opts = getoptx(varargin);
    trjpath = parseopts('trjpath', opts, '/trjdata');

% './trjdata/000/byobj/lnk/####_*'

35 lnkpath = sprintf('.%s%c%03d%cbbyobj%clnk', trjpath, filesep, ...
                    locid, filesep, filesep);
    lnkfile = dir(sprintf('%s%c%04d_*', lnkpath, filesep, objid));
    lnkfile = lnkfile(1);
    lnkfile = [lnkpath filesep lnkfile.name];

40
    lnkdata = importdata(lnkfile);
    imgid = getdatacol('imgid', lnkdata);
    objid = getdatacol('objid', lnkdata);

45 objpath = sprintf('.%s%c%03d%cbbyobj', trjpath, filesep, locid, filesep);

    dataout = [];
    for i = 1:length(imgid)-1,
        curimgid = imgid(i);
50     nxtimgid = imgid(i+1);

        % an objid of -1 indicates a "lost" segment of tracking. skip any
        % object data during this period

55     % an objid of NaN indicates a completely lost object. skip any
        % remaining object data.
        if objid(i) ~= -1 && ~isnan(objid(i)),
            objfile = sprintf('%s%c%04d.obj.dat', objpath, filesep, objid(i));
            curdata = importdata(objfile);

60
            objimgid = getdatacol('imgid', curdata);

            rnginit = find(objimgid >= curimgid, 1, 'first');
            rngstop = find(objimgid <= nxtimgid, 1, 'last') - 1;
65     dataout = [dataout; curdata.data(rnginit:rngstop, :)];
        elseif isnan(objid(i)),
            break;
        end;

70 end;

% % load the last segment of data before the object is lost or the run
% % terminates
% curimgid = imgid(end);
75 % objfile = sprintf('%s%c%04d.obj.dat', objpath, filesep, objid(end-1));
% curdata = importdata(objfile);
% objimgid = getdatacol('imgid', curdata);
%
% rnginit = find(objimgid >= curimgid, 1, 'first');

```

```

80 % dataout = [dataout; curdata.data(rnginit:end, :)];

    tmp.data = dataout;
    tmp.textdata = curdata.textdata;
    tmp.colheaders = curdata.colheaders;
85
    dataout = tmp;

```

```

_____ updateprogress.m _____
function [] = updateprogress(hProg, iCurrImg, nTotImgs, tLoopElaps, varargin)
    opts = getopt('points', opts, [3 10]);
    iMinPtAvg = parseopts('points', opts, [3 10]);
    iMaxPtAvg = iMinPtAvg(2);
5 iMinPtAvg = iMinPtAvg(1);

    if iCurrImg < iMinPtAvg,
        set(hProg, 'name', ...
            sprintf('%d/%d) Calculating Time Remaining ...', ...
10                                     iCurrImg, nTotImgs));
    elseif iCurrImg < iMaxPtAvg && iCurrImg > iMinPtAvg,
        nTimeRemain = ...
            mean(tLoopElaps(end-iCurrImg+2:end))*(nTotImgs - iCurrImg)/60;
        set(hProg, 'name', ...
15         sprintf('%d/%d) Time Remaining: %.2f min', ...
                                     iCurrImg, nTotImgs, nTimeRemain));
    elseif iCurrImg > iMaxPtAvg,
        nTimeRemain = ...
            mean(tLoopElaps(end-iMaxPtAvg+1:end))*(nTotImgs - iCurrImg)/60;
20     set(hProg, 'name', sprintf('%d/%d) Time Remaining: %.2f min', ...
                                     iCurrImg, nTotImgs, nTimeRemain));
    end;

```

C

Hardware Platforms

C.1 Automated microscopy platform

C.1.1 Hardware configuration

The imaging base was a Nikon Diaphot TMD advanced research grade inverted epifluorescence microscope. The microscope had a five position nosepiece configured with 4x 0.1NA Dry, 10x 0.25NA Dry, 20x 0.45NA Dry Ph2, 40x 1.00NA Oil, and 100x 1.40NA Oil imaging objectives and 10x eyepieces. The transmitted light path utilized a 100W HMX-4 halogen lamp, a neutral color balance filter, a green bandpass filter, and a 0.3NA ELWD condenser. The fluorescence light path utilized an EXFO X-Cite 120 mercury halide light source with an attenuation iris, a IR hot mirror and a Cyan/Yellow/Red-fluorescent protein Sedat spectral excitation filter set (Chroma) mounted in motorized filter wheels (Prior Scientific). Both illumination light paths were shuttered at the source with Uniblitz VS35 high speed shutters (Vincent Associates) controlled by a DMM-V4 four channel shutter driver. Image acquisition was performed using a Hamamatsu Orca-ER cooled CCD camera

with an IEEE-1394 firewire interface. The camera was coupled to the microscope via a 1.0x c-mount photocoupler (Diagnostic instruments). Stage translation and fine focus was motorized using a Prior Scientific Proscan-II XY motorized stage and a Standard fine focus drive, respectively. Filter wheels, XY translation, and Z motion were all controlled using a Proscan-II four axis with autofocus controller.

C.1.2 Generalized light paths

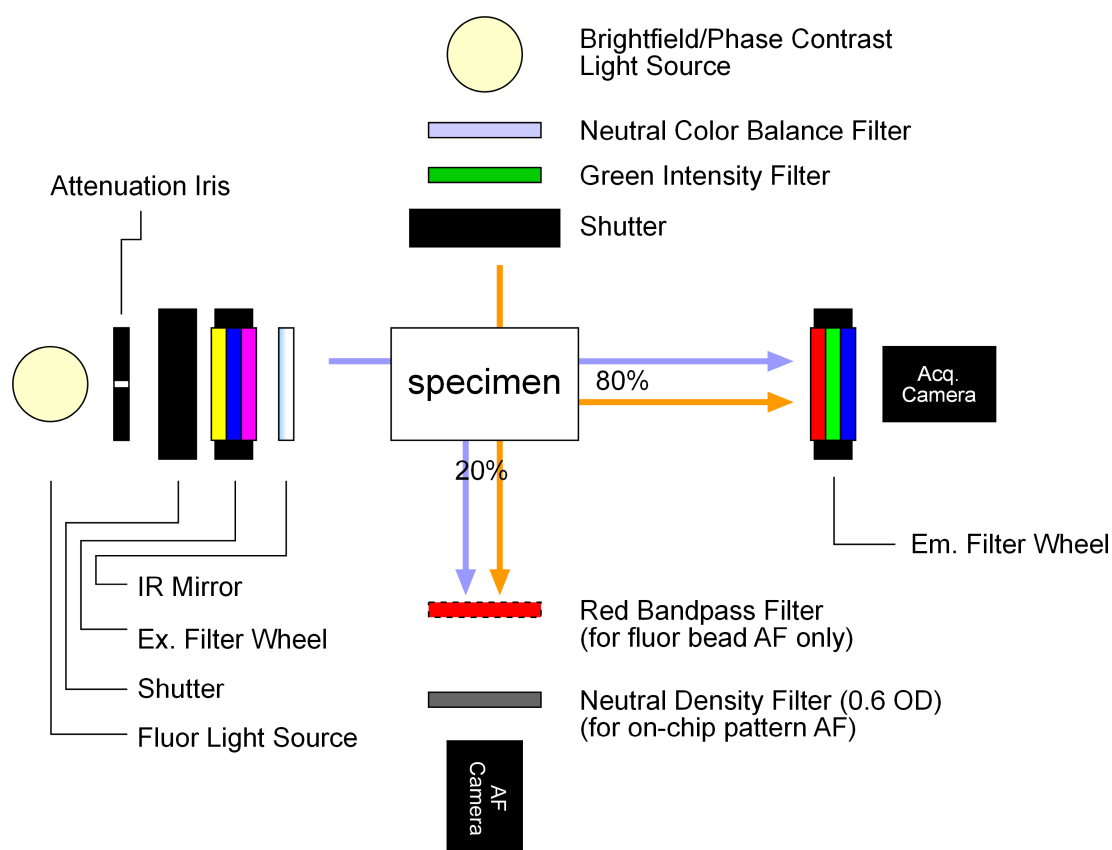


Figure C.1: Imaging optical train and light paths

C.1.3 Imaging algorithm

The imaging system used in this work was controlled using a custom interface developed in National Instruments LabVIEW. To maintain efficiency, the user interface is solely an image collection platform and in this respect its functionality is equivalent to commercially available imaging platforms such as ImagePro (Media Cybernetics), MetaVue (Universal Imaging), or IPLab (Becton Dickinson). The sole purpose of using LabVIEW was the reduction in overall system cost as well as the maintenance of fine tuned system configurability. However, due to the graphical nature of the LabVIEW programming language and the complexity of the interface, only abstract algorithms could be presented here. Complete source code is available online at <http://biodynamics.ucsd.edu>.

Algorithm 1: Microscope operation algorithm

```

Initialize channel stack;
Initialize locations (set pattern);
repeat
  Query acquisition state;
  Check channel indices for remaining images;
  if Acquisition is active and images remain then
    for Each Location do
      Get XYZ coordinates;
      Adjust for rotation;
      Adjust for flatness;
      Goto adjusted XYZ;
      for Each Channel do
        Query acquisition type;
        Check elapsed times for each channel;
        if Acquiring single images
          OR (
            Acquiring an image sequence
            AND the current channel. index is < max channel index
            AND the channel interval has elapsed
          ) then
            Retrieve channel settings;
            Configure hardware for focus;
            Run focus algorithm;
            Retrieve focus results;
            Configure hardware for acquisition;
            Acquire image;
            Log focus results;
            Log acquisition details;
            Increment channel index by 1;
            Reset channel elapsed time to 0;
          endif
        endif
      endfor
    endif
  endfor
endif
until USER-STOP ;

```


C.2 Waveform generation platform

C.2.1 Hardware configuration

The core of the waveform generation system was a pair of card mounted voltage to pressure servo regulators (Bellofram) which operated on a 0–10V signal, and controlled output pressure between 0–1psi accordingly. Voltage signals were generated using a LabVIEW program interfaced with the analog output channels of a PCI-6021 multifunction digital/analog input/output board. The pressure regulators were powered by a common 12VDC source and input pressure supply was regulated to 10psi from house air. Controlled pressure output was passed through a 23ga restriction segment to drop the pressure to within a 0–12inH₂O range and fed to fluid reservoirs. A 20ga bleed orifice was placed in parallel with the reservoir to compensate for the lack of pressure relieving on the servo regulators upstream.

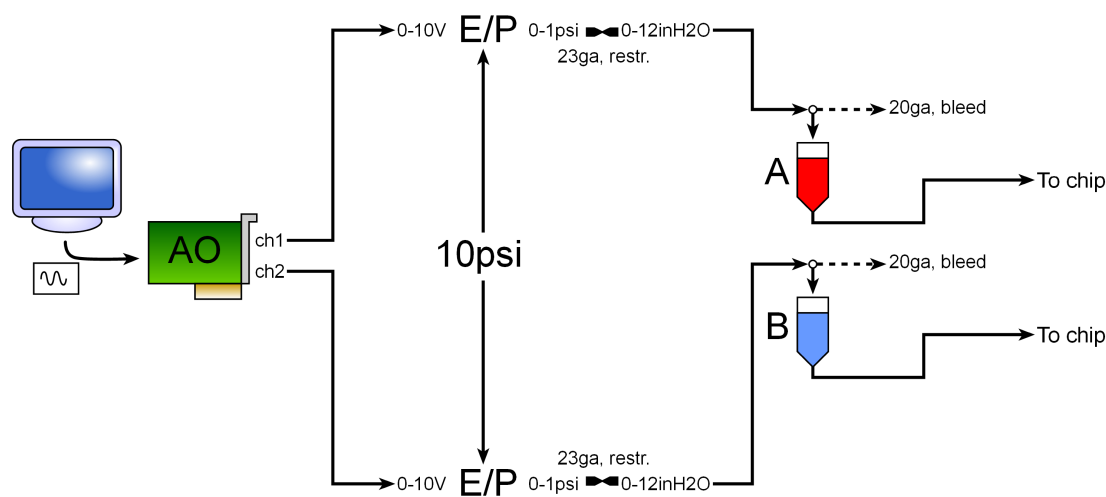
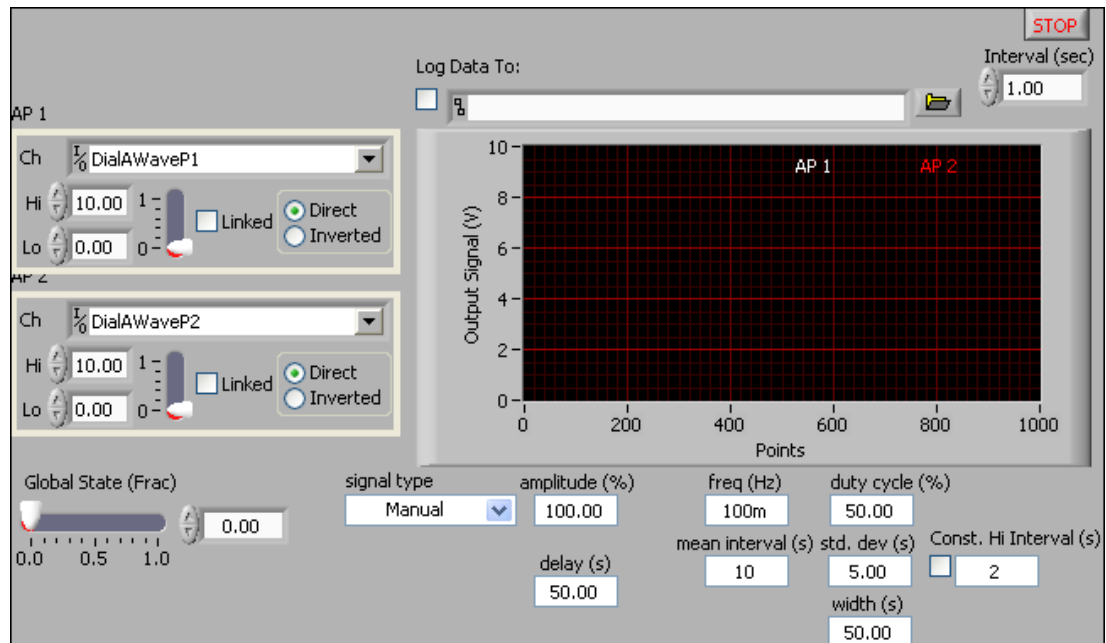


Figure C.2: Waveform generation system configuration

C.2.2 Source documentation

Front Panel



Controls and Indicators



stop



Log Press Interval (sec) Interval between data points in pressure state data log.



Log Press Data? Select this field to start/continue pressure data logging.



Log Press File Location of pressure state log file



freq (Hz) frequency is the frequency of the waveform in units of hertz. The default is 10.



duty cycle (%) square wave duty cycle is the percentage of time a square wave remains high versus low over one period. The VI uses this parameter only if the **signal type** is a square wave. The default is 50%.



std. dev (s) standard deviation is the standard deviation of the generated noise. The default is 1.0.



mean interval (s) average length of time for intervals used by the random step function.



width (s) width for the pulse function.



delay (s) square wave duty cycle is the percentage of time a square wave remains high versus low over one period. The VI uses this parameter only if the **signal type** is a square wave. The default is 50%.



Global State (Frac) Global output state. Controls any linked channels.



amplitude (%) square wave duty cycle is the percentage of time a square wave remains high versus low over one period. The VI uses this parameter only if the **signal type** is a square wave. The default is 50%.



AP 2 Configures output for analog pressure channel 2.



Ch Channel selector for output pressure



Hi Hi voltage value for output pressure control



Lo Lo voltage value for output pressure control



State Output state slider



Linked Output link to global state slider. If selected the channel is controlled by the global state slider. Correspondance to the global state depends on the linkage type.



LinkType Method for linking channel to global state. Direct linkage means that the channel state exactly replicates the global state. Inverted means the channel state is the mirror opposite of the global state.



Direct



Inverted















signal type **signal type** is the type of waveform to generate.



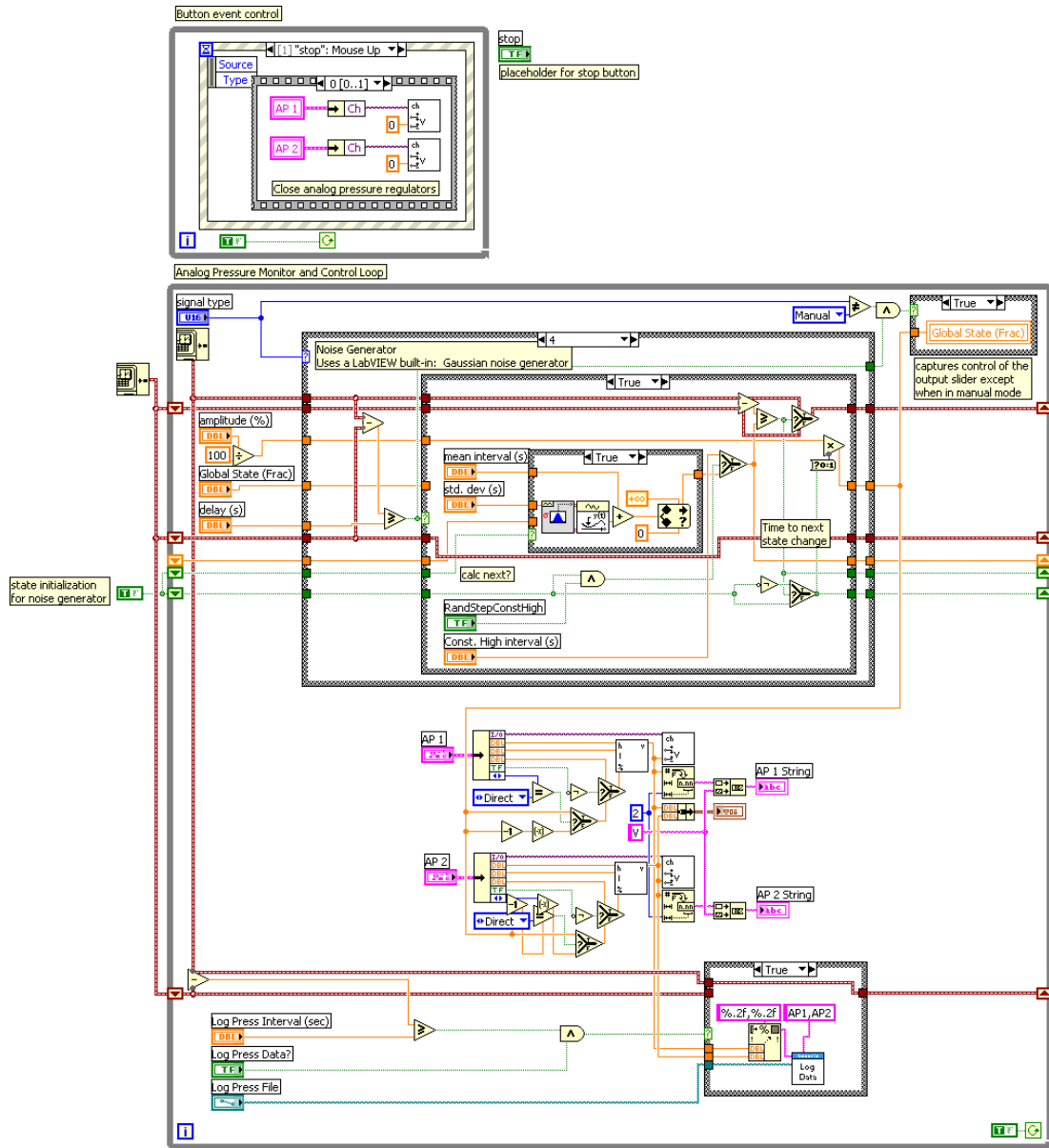
AP 1 Configures output for analog pressure channel 1.



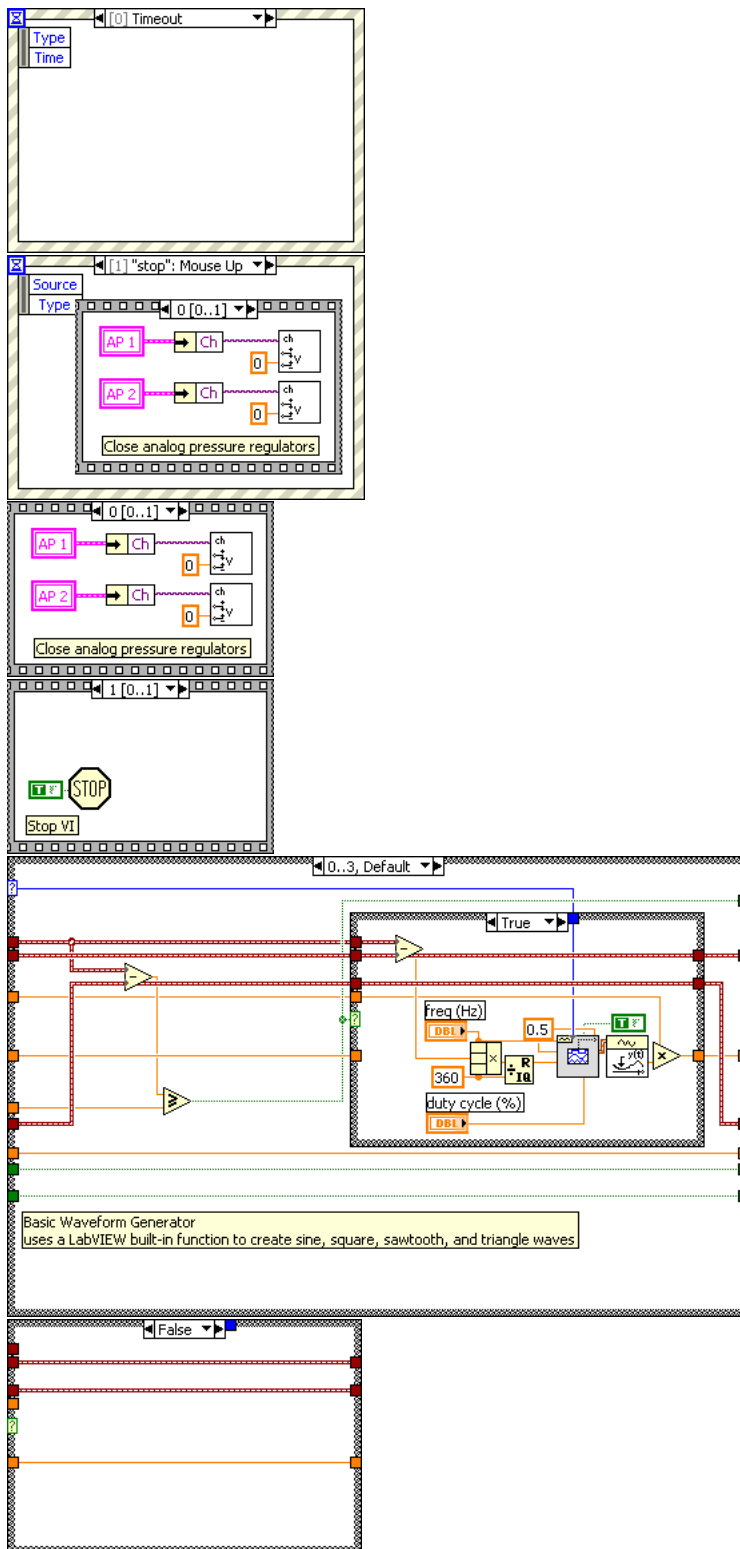
Ch Channel selector for output pressure

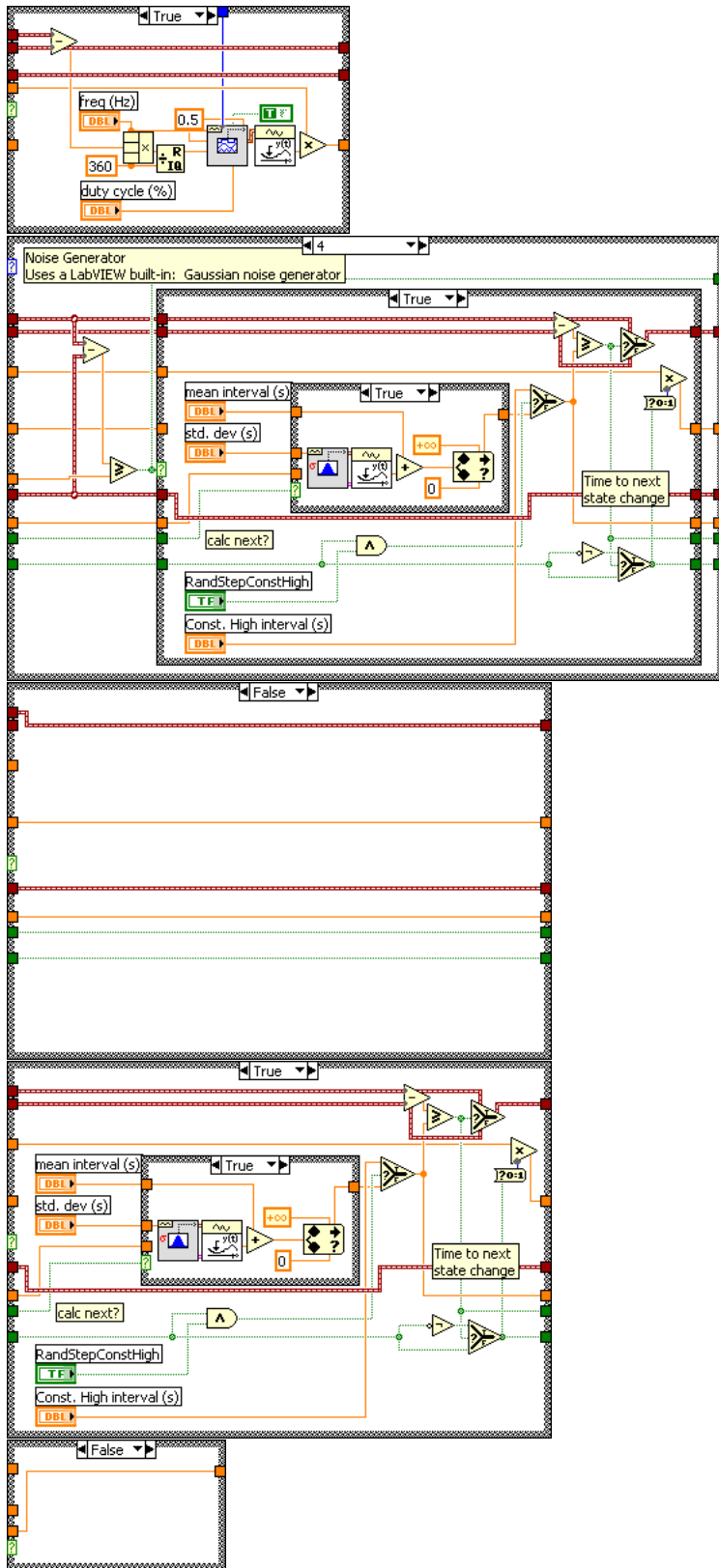
-  **Hi** Hi voltage value for output pressure control
-  **Lo** Lo voltage value for output pressure control
-  **State** Output state slider
-  **Linked** Output link to global state slider. If selected the channel is controlled by the global state slider. Correspondance to the global state depends on the linkage type.
-  **LinkType** Method for linking channel to global state. Direct linkage means that the channel state exactly replicates the global state. Inverted means the channel state is the mirror opposite of the global state.
 -  **Direct**
 -  **Inverted**
-  **RandStepConstHigh**
-  **Const. High interval (s)** Value for constant 'hi' state interval length for random step function
-  **AP 2 String**
-  **AP 1 String**
-  **Analog Pressure Monitor** Graphical display of pressure state for all active channels.

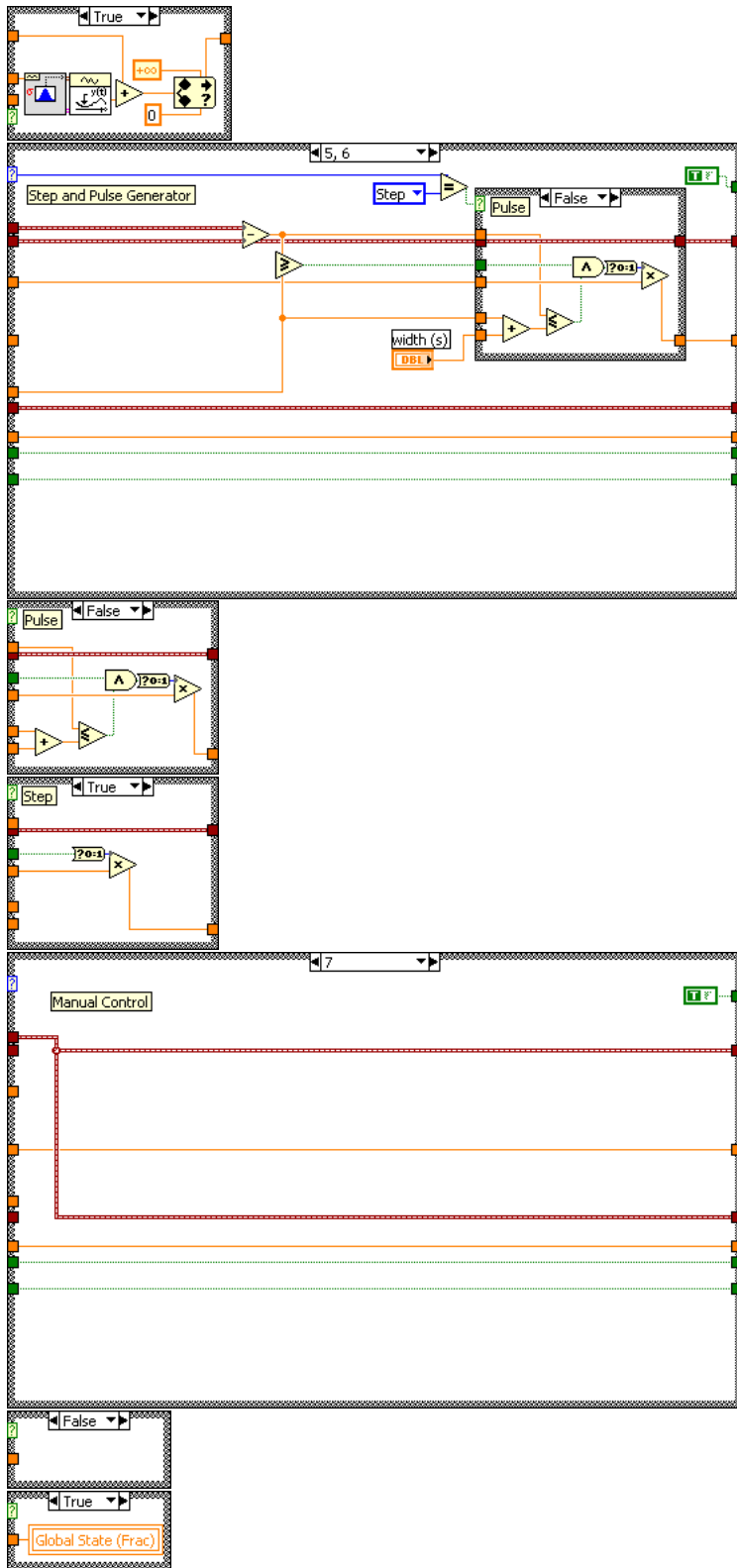
Block Diagram

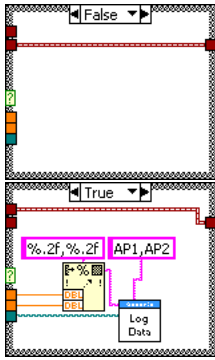


Hierarchical states









D

Microfluidic Devices

D.1 T μ C: Tesla micro-Chemostat

D.1.1 Fabrication

Table D.1: Master mold feature height specifications. [†]Photoresists are SU-8 unless otherwise specified. [‡]These layers were patterned additively (e.g. no development step between this and the prior layer)

Layer	Thickness (μm)	Photoresist [†]	Spin Speed (rpm)
bead trap	2	2002	3000
chamber	4	2002	2000
flow channels	12	2010	2500
access port	38	2015	1000
thermal channel	260	2100	1000

D.1.2 Device Schematic and Port Assignments

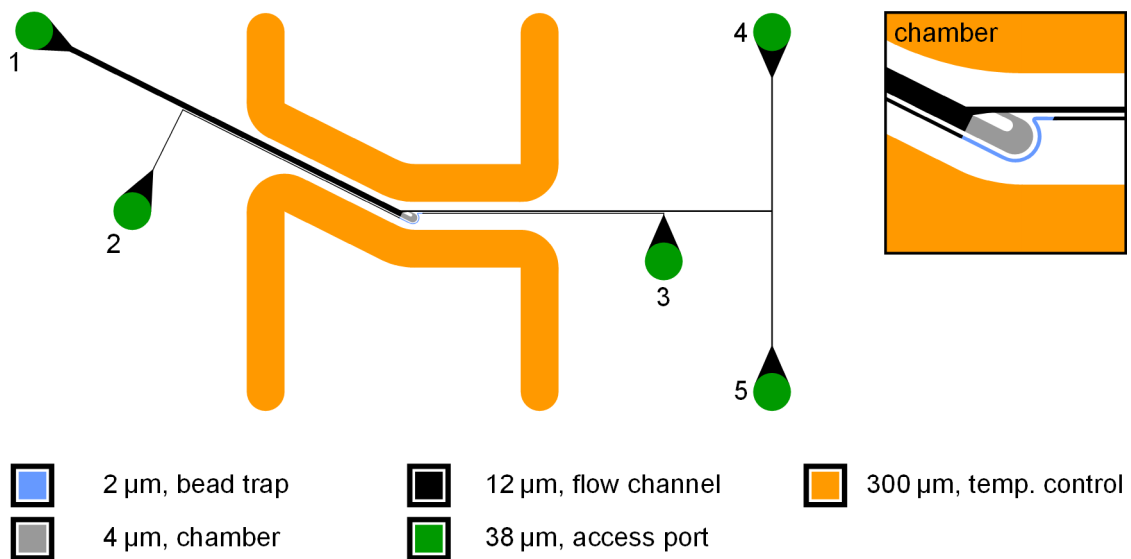


Figure D.1: Device schematic for $T_{\mu}C$. Inset displays a magnified view of the growth chamber.

Table D.2: Port assignments for $T^2_{\mu}C$.

Port	Abbr	Usage
1	C	input, cell suspension
2	Bin	input, af/calibration bead loading
3	Bout	input, af/calibration bead outflow
4	M	input, growth media
5	W	output, common waste

D.2 T² μ C: Temporal Tesla micro-Chemostat

D.2.1 Fabrication

Table D.3: Master mold feature height specifications. [†]Photoresists are SU-8 unless otherwise specified. [‡]These layers were patterned additively (e.g. no development step between this and the prior layer)

Layer	Thickness (μm)	Photoresist [†]	Spin Speed (rpm)
feeding channels	1	2001	3000
chamber	4	2002	2000
af pattern	4	2002	2000
flow network	10	2010	3000
chaotic mixer grooves [‡]	2	2002	3000
thermal channel (1)	50	2050	3000
thermal channel (2) [‡]	260	2100	1000

D.2.2 Device Schematic and Port Assignments

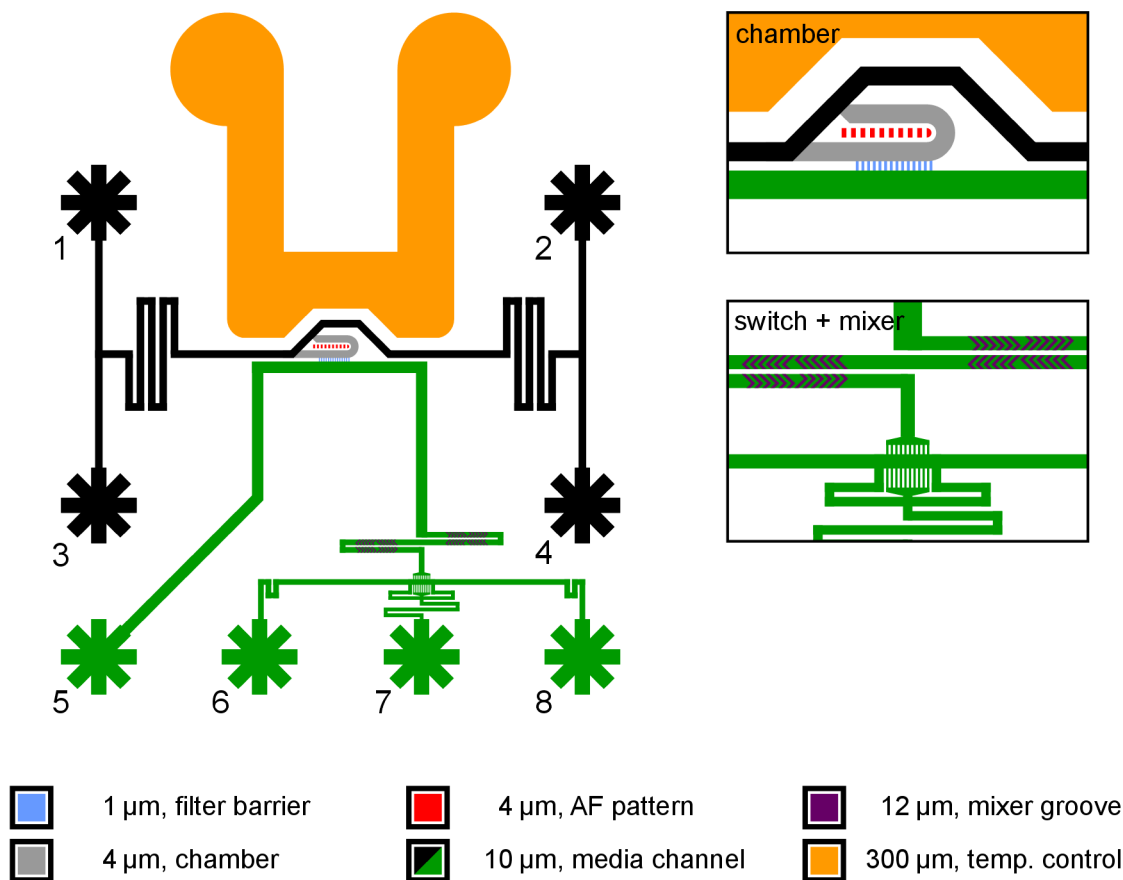


Figure D.2: Device schematic for $T^2\mu C$. Insets display magnified views of the growth chamber and on-chip media switch.

Table D.4: Port assignments for $T^2\mu C$.

Port	Abbr	Usage
1	Wc	output, cell suspension waste
2	Wm	output, loading media waste
3	C	input, cell suspension
4	M	input, loading media (media 0)
5	Ws	output, switching channel waste
6	I	input, media 1 (inducer)
7	S	output, aux. waste for switch
8	B	input, media 2 (blank)

D.2.3 Operation Protocol

Required Materials

- bonded microfluidic device
- microfluidic lines, 8 total
- fluid reservoirs, 8 total
- reservoir pressure top, 8 total
- multichannel pressure controller, 4 channels required, 8 channels recommended
- inverted microscope with imaging camera, shuttered light sources (transmitted and fluorescence), and appropriate acquisition software

optional thermal control fluid lines

optional heated/chilled water circulator

Naming and Media Conventions

There are eight fluidic ports on the device and eight corresponding fluid reservoirs. There are 4 “waste” reservoirs, e.g. where fluid flow exits the device, and 4 “media/input” reservoirs where fluid enters the device. Waste reservoirs are denoted by a “W” label, except for the auxiliary waste port used by the integrated media switch. This port is labeled “S” which stands for “shunt” for historic reasons. The other waste ports are labeled: Ws, Wm, and Wc, for switch waste, loading media waste, and cell suspension waste, respectively. All output reservoirs should contain only sterile filtered water. All input media should be

sterile filtered if possible. This reduces the amount of particulate debris that could block microfluidic channels, rendering a device useless.

The input reservoirs are labeled: M, I, and B, for loading media, “induction” media, and blank media. The “M” reservoir contains unaltered growth medium and under most circumstances should be the same media used to prepare the cell suspension for overnight (normal batch) culture. The “I” reservoir contains growth media with any chemical agents that will elicit a cellular response such as inducers, activators, or repressors. The “B” reservoir should be the same as the media in the “I” reservoir WITHOUT the aforementioned chemical agents, and in most cases is exactly the same as the media in the “M” reservoir. Reservoirs “I” or “B” should also include flow tracing dye (e.g. 0.01 mg/mL sulforhodamine 101, Sigma S7563) which is used to prepare the integrated media switch for operation. For most cases, the “I” reservoir has the additional tracer dye.

The remaining input reservoir is unlabeled and is used for the cell suspension. This is the only reservoir that is not recycled for subsequent experiments. During operation, this reservoir is converted to an output (waste) reservoir to reduce cellular fouling of the device. This will be explained in more detail below.

Protocol

Line Attachment and Device Wetting

1. Secure the device to the microscope stage insert place the device on the microscope (4x or 10x magnification). If available, do not secure rotational movement of the stage insert at this point in time. Inspect the device for any defects that could impair its performance — i.e. channel leakage due to poor bonding, or channel blockage due to

- dust particles. Discard defective devices.
2. Place all reservoirs on the gravity towers and ensure that all the fluid connection lines are sufficiently primed and absent of bubbles. Attach the pressure control lines to the tops of the syringes and set the source pressure on the pressure controller to 5 psi.
 3. Set the heights of the reservoirs to their operational positions. These positions are specific to each experimental setup, but generally they should be (inH₂O): 8, 8, 10, 19.5, 14.5, 20, 20, and 3, for W_m, W_c, W_s, S, M, I, B, Cells, which are the heights specified used in MOCA simulations.
 4. Connect the W_s, W_m, S reservoirs to ports 5, 2, and 7 on the device, respectively, and activate pressure to the reservoirs.
 5. Depending on slight variations in chip manufacturing and chip age, the following may happen in any order. This is normal and does not impair the function of the device.
 - (a) When fluid appears at port 4, attach the M reservoir to it.
 - (b) When fluid appears at port 6, attach the I reservoir to it.
 - (c) When fluid appears at port 8, attach the B reservoir to it.
 6. After a few seconds, fluid should appear at either port 1 or port 3. These two ports are interchangeable as either the entrance for cell suspension (C) or exit for cell waste (W_c) If fluid does not appear at these ports within 5 minutes, pressurize the both the M and W_m reservoirs to aid flow through the device. Once a fluid bead appears attach the W_c reservoir to the corresponding port and turn off any additional pressure. DO NOT attach the cell suspension reservoir at this time.

7. Place the device on the microscope. Inspect all channels for pockets of air. Check to see if the chamber and microchannels that connect it to the integrated media switch are appropriately wetted. If there are air pockets anywhere in the chip, remove them by supplying 5 psi of pressure to ALL attached reservoirs. Once all air pockets have been removed from the device, turn off the external supply pressure.

Media Switch Setup and Operation

1. Move the field of view (4x or 10x magnification) to the media switching region (see device schematic), and illuminate using appropriate fluorescence settings for the tracer dye used. If possible, rotationally square and lock down the device at this point in time.
2. Set the source pressure on the controller to 5 psi.
3. Purge the fluid that may have back flowed into ports 6 and 8 by pressurizing reservoirs I and B, simultaneously. You should see an interface form between the dyed and undyed media midway across the output channel.
4. Monitor the field histogram and note the value of the maximum field intensity. When the maximum stops increasing and the profile remains sufficiently constant, the ports are adequately purged and pressure may be removed.
5. Manually adjust the height of the I and B reservoirs so that the interface between the input streams occurs exactly in the middle of the output stream.
6. If a computer controlled concentration waveform generator is installed skip to step 11.

7. Close all pressure valves and reduce the source pressure to 0 psi. Open the pressure valve to the B reservoir and gradually increase the source pressure until the fluid interface moves from the output channel to one of the flanking overflow bypass channels. Note this pressure as the operational pressure for the switch.
8. Repeat the previous step, but this time pressurizing the I reservoir, to ensure that the switch operation pressure is consistent for both states.
9. Close all pressure valves.
10. Proceed to the cell loading procedures.
11. Replace the pressure connections to the I and B reservoirs with the output lines from the waveform generation system.
12. In the LabVIEW control panel, set the “LO” and “HI” voltages for both analog output channels to 0 volts and 5 volts, respectively.
13. Link both channels to the global control and select one channel to be linked “inverted”. It is recommended that the reservoir without dye (B) be linked in this fashion, thus a 100% output state corresponds to the maximum dye concentration and vice versa. The steps that follow assume this configuration.
14. Set the global state to 0% and adjust the LO value on the I reservoir and the HI Value on the B reservoir so that the flow interface is in the center of the left-hand switch bypass channel.
15. Set the global state to 100% and adjust the HI value on the I reservoir and the LO Value on the B reservoir so that the flow interface is in the center of the right-hand

switch bypass channel.

16. Verify switch operation at 0, 10, 50, 90, and 100% global output states, and make voltage adjustments as necessary. The voltage difference between HI and LO states should NOT be below 1 volt.
17. Once switch operation is verified, set the global output to the desired initial conditions and proceed to the cell loading procedures.

Cell Loading

1. Attach the cell suspension reservoir to the remaining port on the device. Make sure that there is a bead of water at the port before attaching the reservoir. If not, briefly pressurize the Wc reservoir until one appears.
2. Illuminate the device with transmitted light and move to a field of view that displays the chamber entrance closest to the feeding channels.
3. Set the source pressure to 3 psi and pressurize both the cell suspension (C) and loading media (M) reservoirs. This will inject cells into the loading segment of the chip. Note, if you observe the field using the fluorescent settings for your tracer dye, you should be able to see laminar interface between fluorescent fluid and non-fluorescent fluid within the region. Increase the supply pressure so that this interface is positioned at the entrance to the chamber.
4. Cells should now start to flow into the trapping loop/growth chamber. If cells are retained at the entrance, you may increase the pressure up to 10 psi and also gently flick the fluidic lines to help push cells in.

5. Once a satisfactory number of cells have entered the chamber, TURN OFF pressure to the cell suspension ONLY. This converts cell suspension reservoir to a waste reservoir because of its base height relative to all the other reservoirs. Loading media will now purge the loading segment to reduce device fouling. If cells remain at the entrance to the growth chamber, gently flick the microfluidic lines to dislodge them. This will also help to cluster the cells within the chamber.
6. TURN OFF pressure to the M reservoir. Cells should now begin to distribute throughout the chamber. If cells do not distribute you can gently flick the microfluidic lines to get them moving. If you accidentally purge the chamber of cells, you can repeat the above 3 steps.
7. If a computer controlled concentration waveform generator is installed you may skip the next step.
8. Once cells are loaded and satisfactorily distributed, set the source pressure to the operational value for the switch and TURN ON pressure to the either the B or I reservoir, which ever media you wish to initialize the experiment with.

Data Acquisition

1. Secure the microfluidic lines to the stage. Be sure to provide enough slack for any required stage movements.
2. Change to the desired magnification (typically 40x) and move to the desired field of view. If a motorized stage is installed, prepare the operating software appropriately. Typically this requires setting the stage origin and/or defining a scan pattern.

3. Acquire using your desired acquisition settings. When appropriate, change the state of the media using the pressure valves for reservoirs I and B or a computer controlled concentration waveform generator.

Cleanup

1. Turn off all pressure valves.
2. Remove the device from the stage insert.
3. Remove the fluidic lines from the device by holding the device by the PDMS portion and pulling firmly on the lines.
4. Collect the lines together with a binder clip and place them so that all flow is collected in a large waste collection beaker.
5. Dump the contents of each reservoir into the waste collection beaker and fill each one with 0.22 μm filtered water, replacing the pressure tops when done.
6. Set the pressure source to 15 psi and open all pressure valves. This will flush the lines with fluid.
7. Repeat the previous two steps and additional three times, once more with water, followed with 50% isopropanol/water, followed by water once again. Do not drain the lines completely on the last flush.
8. With the lines still primed with fluid, place them in a glass beaker in an ultrasonic bath for 10–15 minutes. During this time, rinse out the reservoirs and pressure tops with de-ionized water.

9. Reattach the lines to the reservoirs and flush once more with filtered water. Allow the reservoirs to fully drain and air to purge remaining fluid from the lines.
10. Let the lines dry under forced air flow for about 10–15 minutes.
11. Detach the lines from the reservoirs and remove the pressure tops. Place all items in sterilization pouches, one for lines, one for reservoirs, and one for pressure tops. Autoclave on a short wrapped dry cycle (15 minute sterilization time) and store in a dry location for future use.

D.3 Glial Network Stimulator

D.3.1 Fabrication

Table D.5: Master mold feature height specifications. †Photoresists are SU-8 unless otherwise specified.

Layer	Thickness (μm)	Photoresist [†]	Spin Speed (rpm)
filter channels	1	2001	3000
perfusion channels	10	2010	3000
stimulant channels			
chamber squeeze-thru barrier			
chamber/loading channels	50	2050	3000

D.3.2 Device Schematic and Port Assignments

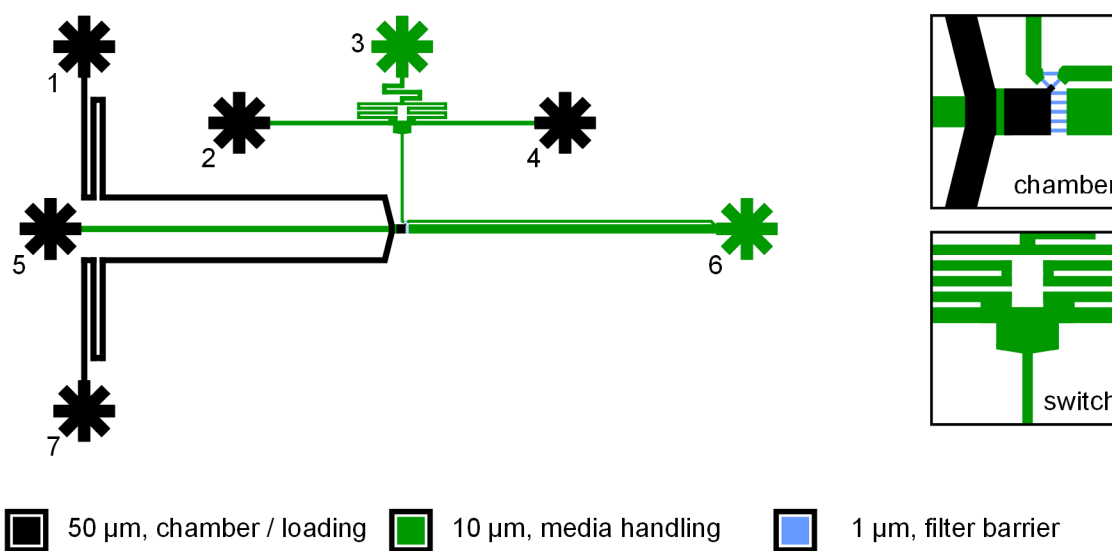


Figure D.3: Device schematic for glial network stimulation device. Insets display magnified views of the growth chamber and on-chip media switch.

Table D.6: Port assignments for the glial network stimulation device.

Port	Abbr	Usage
1	C	input, cell suspension
2	I	input, media 1 (inducer)
3	S	output, aux. waste for switch
4	B	input, media 2 (blank)
5	M	input, primary perfusion media
6	Ws	output, chamber + stimulant outflow waste
7	Wc	output, cell suspension + overflow perfusion waste

D.4 DynaGrad: Dynamic Chemical Gradient Device

D.4.1 Fabrication

Table D.7: Master mold feature height specifications. †Photoresists are SU-8 unless otherwise specified. ‡ Bacterial and yeast devices require patterning the gradient outflow channel with adhesion molecules such as polylysine (bacterial) or Conavalin-A (yeast).

Layer	Thickness (μm)	Photoresist [†]	Spin Speed (rpm)
bacterial device [‡]	6	2005	2500
yeast device [‡]	10	2010	3000
mammalian device	50	2050	3000

D.4.2 Device Schematic and Port Assignments

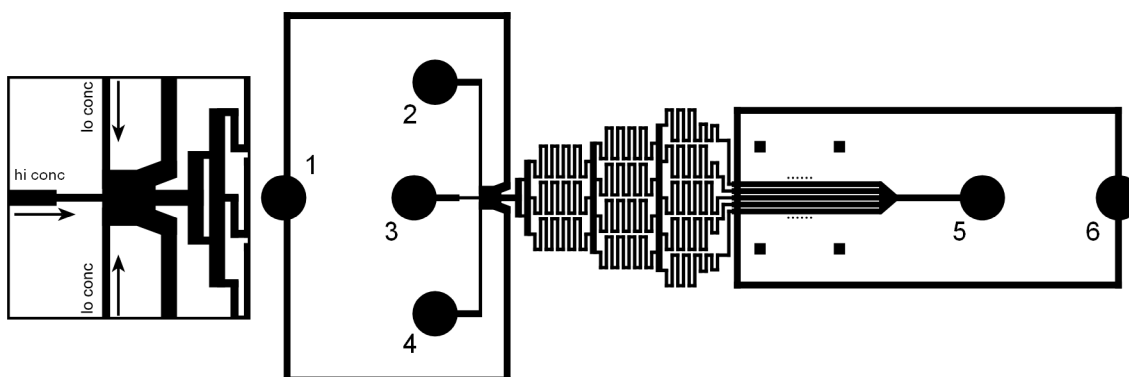


Figure D.4: Device schematic for dynamic gradient device. Inset displays magnified views of the on-chip media switch.

Table D.8: Port assignments for the dynamic gradient device.

Port	Abbr	Usage
1	S	output, aux. waste for switch
2	M3	input, low concentration media (blank)
3	M2	input, high concentration media (inducer)
4	M1	input, low concentration media (blank)
5	W	output, common waste port
6	C	input, cell suspension loading port

E

Computational code for $T^2\mu\text{C}$ data simulations and analysis

E.1 Data analysis

The experimental analysis quantifies blob fluorescence trajectories. A blob is the combination of all identified objects within the field of view and analysis. Blob area is the sum of all subobject areas, while measured mean fluorescence is the average of all subobject values. All fluorescence trajectories are detrended prior to extracting amplitude and phase information. Analysis of simulation data proceeds in a similar fashion.

```
function t = chartime(k, n)
t = log(n)./k;
```

```
function [dataout, datalg, datasm] = detrend(datain, winlg, winsm, method)
datalg = smooth(datain, floor(winlg), method);
datasm = smooth(datain, floor(winsm), method);
dataout = datasm - datalg;
```

```
function exportfig(h, name, fmts)
% EXPORTFIG(h, name, fmts)
```

```

%
% Saves figure specified by handle 'h' with name 'name' in formats
5 % specified in the 1D cell array of strings 'fmts'.
%
% See Also:
%   saveas
if iscell(fmts),
10   for i=1:length(fmts),
       saveas(h, name, fmts{i});
   end;
end;

```

```

_____ getdatacol.m _____
function [datavec] = getdatacol(sCol, Data, varargin)
%GETDATACOL
% [datavec] = getdatacol(sCol, Data, cols)
%
5 % Gets data from a column in a data set
if nargin == 3,
    cols = varargin{1};
elseif nargin == 2,
    if isstruct(Data),
10     cols = Data.colheaders;
        Data = Data.data;
    else,
        error(['invalid input: '...
              'Data must be a structure or column '...
15     'headers must be defined']);
    end;
end;

datavec = Data(:, strmatch(upper(sCol), strtrim(upper(cols)), 'exact'));

```

```

_____ normrange.m _____
function ndata=normrange(data)
ndata = (data - min(data(:)))./(max(data(:)) - min(data(:)));

```

```

_____ ttmcblob.m _____
% blob data analysis of frequency runs
% runs with 2/0.2/(0.25)% r/g/(d):
clear, caf%, clc

5 % script to define experiments
ttmc_expts
expts =expts(strmatch('k699',{expts{:},5}), :);
%expts = expts(find([expts{:},1] == 2.25),:);

10 bDisplayRaw = true;      % display the raw data
    bDisplayRawDT = true;  % display the detrended raw data

    bDisplayPfAmp = true;  % display amplitude analysis process figures
    bDisplayPfPhi = true;  % display phase analysis process figures
15 cDisplayPf = {};
    if bDisplayPfAmp, cDisplayPf = {cDisplayPf{:}, 'amp'}; end;
    if bDisplayPfPhi, cDisplayPf = {cDisplayPf{:}, 'phi'}; end;

    bSummarize = true;     % display summary plots e.g. amp vs frq

```

```

20 bVerbose = false;      % generate detailed text output in command window
   bVerboseShort = true; % generate short text output in command window

   bPrint = false;      % print all generated figures
   bSave = true;        % save all generated figures
25 cFormats = {'fig', 'png'}; % output formats for the 'save' option
   cOutput = {};
   if bPrint, cOutput = {cOutput{:}}, 'print'}; end;
   if bSave, cOutput = {cOutput{:}}, 'save'}; end;

30 tdoub = [];
   rsq = [];
   per = [];
   frq = [];
   amp = [];
35 phi = [];

   for iExpt = 1:size(expts, 1),
       stExpt = struct(...
           'id',          expts{iExpt, 3}(4:end), ...
40         'path',       expts{iExpt, 3}, ...
           'strain',     expts{iExpt, 5}, ...
           'channel',    expts{iExpt, 6}, ...
           'period',     expts{iExpt, 1}, ...
           'interval',   expts{iExpt, 2} ...
45         );
       stExpt.figname = sprintf('%s_%5.3f_%s', ...
                               stExpt.strain, stExpt.period, stExpt.id);
       stExpt.figname = strrep(stExpt.figname, '.', 'p');
       stExpt.figname = strrep(stExpt.figname, '/', '-');

50       sExptID = expts{iExpt, 3}(4:end);
       sStrain = expts{iExpt, 5};
       iQntCh = expts{iExpt, 6};
       nPeriod = expts{iExpt, 1};      % hrs
55       nInterval = expts{iExpt, 2};  % min

       sFigName = sprintf('%s_%5.3f_%s', sStrain, nPeriod, sExptID);
       sFigName = strrep(sFigName, '.', 'p');
       sFigName = strrep(sFigName, '/', '-');

60       if bVerbose, fprintf('Period:\t\t\t%5.2f\n', stExpt.period); end;

       per(iExpt) = stExpt.period; % hr
       frq(iExpt) = 2*pi/stExpt.period; % rads/hr

65       % retrieve the data from the blob output file
       sBlobDataFile = sprintf('%s/%s/%s', ...
                               stExpt.path, 'objdata', '000.blob.dat');
       data = importdata(sBlobDataFile);
70       t = getdatacol('imgid', data)*stExpt.interval/60;
       f = getdatacol(['mean' num2str(stExpt.channel)], data); % blob fluor
       b = getdatacol(['bglvl' num2str(stExpt.channel)], data); % blob bg lvl
       r = getdatacol('bglvl1', data); % input signal
       a = getdatacol('area', data); % blob area

75       % detrend/smooth the data

```

```

nPtsPerPeriod = stExpt.period*60/stExpt.interval; % num pts in a period

winlg = stExpt.period*60/stExpt.interval*3; % num data pts, lg interval
80 %winsm = nPeriod*60/nInterval/3; % num data points, small interval
winxsm = 3; % num data points, extra small interval
winsm = 12; % num data points, small interval
winlgarea = 48; % this is constant for area traces

85 if winsm > nPtsPerPeriod, winsm = winxsm; end;

[asdt, aslg, assm] = detrend(a, winlgarea, winsm, 'rlowess');
[fsdt, fslg, fssm] = detrend(f, winlg, winsm, 'rlowess');
[rsdt, rslg, rssm] = detrend(r, winlg, winsm, 'rlowess');
90 [bsdt, bslg, bssm] = detrend(b, winlg, winsm, 'rlowess');

if bDisplayRaw,
    figRawRaw = figure;

95     figure(figRawRaw),
        subplot(311), plot(t, f, 'b.',...
                            t, fssm, 'g',...
                            t, fslg, 'r:')
                            ylabel('<FL>')
100    title(sprintf('%s Period (Frequency):%5.2fhr (%5.4fhr^-1)', ...
                    stExpt.id, stExpt.period, 1/stExpt.period));
        figure(figRawRaw),
        subplot(312), plot(t, assm, 'b', t, aslg, 'r:'), ylabel('Area')
        figure(figRawRaw),
105    subplot(313), plot(t, rssm, 'b'), ylabel('glc (<FL>')
        xlabel('time (hr)');
        set(figRawRaw, 'name', sprintf('RAW: %5.3f (%s)', ...
                    stExpt.period, stExpt.id));
        if bPrint,
110            drawnow;
            print(figRawRaw);
        end;
        if bSave,
            drawnow;
115            exportfig(figRawRaw, [stExpt.figname '__raw'], {'fig', 'png'});
        end;
    end;

end;

% use the area trace to determine when the data becomes invalid due to
120 % increased segmentation error (coincides with chamber confluence)
% truncate data beyond this point

%idxend = find(diff(aslg)./diff(t) <= 0, 1)+1; % by derivative (buggy)
idxend = find(assm >= 0.99*max(assm), 1); % by abs max value

125 if isempty(idxend), idxend = length(a); end;

t = t(1:idxend);
asdt = asdt(1:idxend); aslg = aslg(1:idxend); assm = assm(1:idxend);
130 fsdt = fsdt(1:idxend); fslg = fslg(1:idxend); fssm = fssm(1:idxend);
rsdt = rsdt(1:idxend); rslg = rslg(1:idxend); rssm = rssm(1:idxend);
bsdt = bsdt(1:idxend); bslg = bslg(1:idxend); bssm = bssm(1:idxend);

```

```

% fit the area growth and get the growth rate for scaling
135 [res, gof, out] = fit(t, aslg, 'expl');
tdoub(iExpt) = chartime(res.b, 2);
rsq(iExpt) = gof.rsquare;

% normalize and flip the input signal (since glucose is traced but
140 % response is 'produced' by galactose
sigout = 1-normrange(rssm)-0.5;

% compensate for the background in the detrended response
rspout = (fsdt - bsdt)./fslg;%*res.b;
145

% -----
% from here on, the only variables needed are t, rspout, sigout, and
% expt details
% -----

150 % exclude the first period in the data ... which contains artifacts of
% the detrending process
rng = excludedata(t, sigout, 'domain', [0 stExpt.period]);

155 t = t(rng);
sigout = sigout(rng);
rspout = rspout(rng);

if bDisplayRawDT,
160     figRawDT = figure;

     figure(figRawDT),
     subplot(211), plot(t, rspout, 'b'),%, t, rspoutsm, 'r:'),
     ylabel('Detrended <FL>')
165     title(sprintf('%s Period (Frequency):%5.2fhr (%5.4fhr^-1)', ...
         stExpt.id, stExpt.period, 1/stExpt.period));
     figure(figRawDT),
     subplot(212), plot(t, sigout), ylabel('1 - n.glc')
     xlabel('time (hr)');
170     set(figRawDT, 'name', sprintf('RAW DT: %5.3f (%s)', ...
         stExpt.period, stExpt.id));

     if bPrint,
         drawnow;
         print(figRawDT);
175     end;
     if bSave,
         drawnow;
         exportfig(figRawDT, [stExpt.figname '__detrend'], {'fig', 'png'});
     end;
180 end;

[pfamp, pfphi] = ttmc_analysis_peakfind(...
185     stExpt, t, sigout, rspout, ...
     'verbose', bVerbose, ...
     'display', cDisplayPf, ...
     'output', cOutput);

amp(iExpt) = mean(pfamp);
190 amperr(iExpt) = sqrt(var(pfamp));

```

```

    phi(iExpt) = mean(pfphi);
    phierr(iExpt) = sqrt(var(pfphi));

    if bVerboseShort && iExpt == 1,
195     fprintf('%15s%10s%10s%10s%10s%10s\n', ...
            'ExptID', 'Period', 'tdoub', 'rsq', 'amp', 'phi');
    end;
    if bVerboseShort,
        fprintf('%15s%10.4f%10.4f%10.4f%10.4f%10.4f\n', ...
200         stExpt.id, stExpt.period, tdoub(iExpt), rsq(iExpt), ...
            amp(iExpt), phi(iExpt));
    end;

    if bVerbose, fprintf('\n'); end;
205 end;

    if bSummarize,
        % The results:
        % sort the amps and phis by their corresponding frequency value
210     [frq, ixsrts] = sort(frq);

        amp = amp(ixsrts);
        if exist('amperr', 'var'),
            amperr = amperr(ixsrts);
215     else
            amperr = [];
        end;
        phi = phi(ixsrts);
        if exist('phierr', 'var'),
220         phierr = phierr(ixsrts);
        else
            phierr = [];
        end;

225     % generate the summary plots
        ttmc_analysis_summarize(...
            stExpt.strain, frq, amp, amperr, phi, phierr, ...
            'output', cOutput, ...
            'formats', cFormats);
230     end; % if bSummarize

    % save variables for later analysis/plotting
235 save([stExpt.strain '__cfp__amp-pkf_phi-pkf.mat'], ...
        'frq', 'amp', 'amperr', 'phi', 'phierr', 'tdoub', 'rsq')
    close all hidden

```

```

_____ ttmcsim.m _____
function ttmcsim(sStrain, nTDoub)
% ttmcsim
% clear, caf%, clc

5 % options
bDisplayRaw = true;    % display the raw data
bDisplayRawDT = true; % display the detrended raw data

```

```

    bDisplayPfAmp = true;    % display amplitude analysis process figures
10 bDisplayPfPhi = true;    % display phase analysis process figures
    cDisplayPf = {};
    if bDisplayPfAmp, cDisplayPf = {cDisplayPf{:}, 'amp'}; end;
    if bDisplayPfPhi, cDisplayPf = {cDisplayPf{:}, 'phi'}; end;

15 bSummarize = true;      % display summary plots e.g. amp vs frq
    bVerbose = false;      % generate detailed text output in command window
    bVerboseShort = true;  % generate short text output in command window

    bPrint = false;        % print all generated figures
20 bSave = true;           % save all generated figures
    cFormats = {'fig', 'png'}; % output formats for the 'save' option
    cOutput = {};
    if bPrint, cOutput = {cOutput{:}, 'print'}; end;
    if bSave, cOutput = {cOutput{:}, 'save'}; end;

25
    % load the data
    % nTDoub = 4;           % hrs

    % get the files to load:
30 sFileNameTpl = sprintf('TD%.2f_P*', nTDoub);
    sFileNameTpl = strrep(sFileNameTpl, '.', 'p');
    stFiles = dir(['./' sFileNameTpl '.mat']);
    csFiles = {stFiles.name}';

35 for iExpt = 1:length(csFiles),
    % get response data
    data = load(csFiles{iExpt}, ...
        'T', 'gallout', 'gal2out', ...
        'gluout', 'sSimID', 'nPeriod', 'nInterval');

40
    stExpt = struct(...
        'id',          data.sSimID, ...
        'path',        '', ...
        'strain',      sStrain, ...
45        'channel',   1, ... % left over from expt script, for compat
        'period',     data.nPeriod, ...
        'interval',   data.nInterval ...
    );

50
    % process the textual expt identification data to a string that can be
    % used for figures titles and file names.
    stExpt.figname = sprintf('%s_%5.3f_%s', stExpt.strain, ...
        stExpt.period, stExpt.id);
    stExpt.figname = strrep(stExpt.figname, '.', 'p');
55
    stExpt.figname = strrep(stExpt.figname, '/', '-');

    per(iExpt) = stExpt.period; % hr
    frq(iExpt) = 2*pi/stExpt.period; % rads/hr

60
    % get response data in units of hours
    t = data.T/60;

    % truncate the data to anything that occurs within 36hrs
    validx = (t <= 36);
65
    t = t(validx);

```



```

f = data.gallout(validx);
r = data.gluout(validx);
b = zeros(size(f));           % placeholder for background fluor
70
% detrend/smooth the data
nPtsPerPeriod = stExpt.period*60/stExpt.interval; % num pts in a period

winlg = stExpt.period*60/stExpt.interval*3; % num data pts, lg interval
75
% small smoothing intervals are set to 1 for simulation data since it
% is already smooth.
winxsm = 1;           % num data points, extra small interval
winsm = 1;           % num data points, small interval
80
if winsm > nPtsPerPeriod, winsm = winxsm; end;

[fsdt, fslg, fssm] = detrend(f, winlg, winsm, 'rloess');
[rsdt, rslg, rssm] = detrend(r, winlg, winsm, 'rloess');
85
if bDisplayRaw,
    figRawRaw = figure;

    figure(figRawRaw),
90    subplot(211),
        plot(t, f, 'b.', ...
            t, fssm, 'g', ...
            t, fslg, 'r:'),
        ylabel('<FL>')
95    title( ...
        sprintf('%s Period (Frequency):%5.2fhr (%5.4fhr^-1)',...
            stExpt.id, ...
            stExpt.period, ...
            1/stExpt.period), ...
        'interpreter', 'none');
100    figure(figRawRaw),
        subplot(212),
            plot(t, rssm, 'b'), ylabel('glc (<FL>')
            xlabel('time (hr)');
105    set(figRawRaw, 'name', ...
        sprintf('RAW: %5.3f (%s)', stExpt.period, stExpt.id));
    if bPrint,
        drawnow;
        print(figRawRaw);
110    end;
    if bSave,
        drawnow;
        exportfig(figRawRaw, [stExpt.figname '__raw'], {'fig', 'png'});
    end;
115 end;

% normalize and flip the input signal (since glucose is traced but
% response is 'produced' by galactose
sigout = 1-normrange(rssm)-0.5;
120
% compensate for the background in the detrended response
rspout = (fsdt - b)./fslg;%*res.b;

```

```

% run analysis
125 % exclude the first period in the data ... which contains artifacts of
% the detrending process
rng = excludedata(t, sigout, 'domain', [0 stExpt.period]);

t = t(rng);
130 sigout = sigout(rng);
rspout = rspout(rng);

if bDisplayRawDT,
    figRawDT = figure;
135
    figure(figRawDT),
    subplot(211), plot(t, rspout, 'b'), ylabel('Detrended <FL>')
    title(sprintf('%s Period (Frequency):%5.2fhr (%5.4fhr^-1)', ...
        stExpt.id, stExpt.period, 1/stExpt.period), ...
140         'interpreter', 'none');
    figure(figRawDT), subplot(212),
    plot(t, sigout), ylabel('1 - n.glc')
    xlabel('time (hr)');
    set(figRawDT, 'name', sprintf('RAW DT: %5.3f (%s)', ...
145         stExpt.period, stExpt.id));
    if bPrint,
        drawnow;
        print(figRawDT);
    end;
150    if bSave,
        drawnow;
        exportfig(figRawDT, [stExpt.figname '__detrend'], {'fig', 'png'});
    end;
end;

155 [pfamp, pfphi] = ttmc_analysis_peakfind(...
        stExpt, t, sigout, rspout, ...
        'verbose', bVerbose, ...
        'display', cDisplayPf, ...
160        'output', cOutput, ...
        'formats', cFormats);

amp(iExpt) = mean(pfamp);
amperr(iExpt) = sqrt(var(pfamp));
165 phi(iExpt) = mean(pfphi);
phierr(iExpt) = sqrt(var(pfphi));

if bVerboseShort && iExpt == 1,
    fprintf('%20s%10s%10s%10s%10s\n', ...
170         'ExptID', 'Period', 'tdoub', 'amp', 'phi');
end;
if bVerboseShort,
    fprintf('%20s%10.4f%10.4f%10.4f%10.4f\n', ...
175         stExpt.id, stExpt.period, nTDoub, amp(iExpt), phi(iExpt));
end;

end;

if bSummarize,

```

```

180 % The results:
    % sort the amps and phis by their corresponding frequency value
    [frq, ixsr] = sort(frq);

    amp = amp(ixsr);
185 if exist('amperr', 'var'),
        amperr = amperr(ixsr);
    else
        amperr = [];
    end;
190 phi = phi(ixsr);
    if exist('phierr', 'var'),
        phierr = phierr(ixsr);
    else
        phierr = [];
195 end;

    % generate the summary plots
    ttmc_analysis_summarize(...
        stExpt.strain, frq, amp, amperr, phi, phierr, ...
200 'output', cOutput, ...
        'formats', cFormats);

    end; % if bSummarize

205 % set(get(0, 'children'), 'windowstyle', 'docked');
    save([stExpt.strain '__gallp__amp-pkf_phi-pkf.mat'], ...
        'frq', 'amp', 'amperr', 'phi', 'phierr', 'nTDoub')
    close all hidden

```

```

_____ ttmc_analysis_peakfind.m _____
function [ampf, phipf]=ttmc_analysis_peakfind(stExpt, ...
                                                t, sigout, rspout, ...
                                                varargin)

    opts = getopt(varargin);
5 bVerbose = parseopts('verbose', opts, false);

    cDisp = parseopts('display', opts, {});
    if iscell(cDisp) && ~isempty(cDisp),
        bDisplayPfAmp = ~isempty(strmatch('amp', lower(cDisp), 'exact'));
10    bDisplayPfPhi = ~isempty(strmatch('phi', lower(cDisp), 'exact'));

    cOutput = parseopts('output', opts, {});
    if iscell(cOutput) && ~isempty(cOutput),
        bPrint = ~isempty(strmatch('print', lower(cOutput), 'exact'));
15    bSave = ~isempty(strmatch('save', lower(cOutput), 'exact'));
    else
        bPrint = false;
        bSave = false;
    end;
20
    if bSave,
        % only process the formats option if 'save' is specified. no
        % initialization is ok to do since everything is behind the bSave
        % boolean
25    cFormats = parseopts('formats', opts, {'fig', 'png'});
        if ~iscell(cFormats) || isempty(cFormats),

```

```

        % just in case users specify something weird resort to defaults
        cFormats = {'fig', 'png'};
    end;
30 end;

else
    bDisplayPfAmp = false;
    bDisplayPfPhi = false;
35 end;

% -----
% PEAK FINDING CODE
% -----
40

% determine the sensitivity needed for finding local extrema. do this
% by gradually decreasing the sensitivity until there are at least
% three peaks to analyze in both the maximums and minimums. this is
% only needed for the response data since the signal input data is
45 % generally much cleaner.

sens = 0.1;

% find the local extrema in the response data
50 [rspmax rspmin] = peakdet(rspout, (max(rspout) - min(rspout))*sens);
while min([length(rspmax) length(rspmin)]) < 3 && sens >= 0.001,
    sens = sens - 0.001;
    [rspmax rspmin] = peakdet(rspout, (max(rspout) - min(rspout))*sens);
end;
55

% find the local extrema in the input signal data
[sigmax sigmin] = peakdet(sigout, (max(sigout) - min(sigout))*0.01);

60 % AMPLITUDE DETERMINATION
% generate nearest neighbor interpolations. this surrounds each
% extrema with equivalent constant values, generating a step like
% trace.
rspmaxitp = interp1(t(rspmax(:,1)), rspmax(:,2), t, 'nearest');
65 rspminitp = interp1(t(rspmin(:,1)), rspmin(:,2), t, 'nearest');

% each trace is bordered by NaN's extend the numeric values at the ends
rspmaxitpidx = find(~isnan(rspmaxitp));
rspminitpidx = find(~isnan(rspminitp));
70

rspmaxitp(1:rspmaxitpidx(1)) = rspmaxitp(rspmaxitpidx(1));
rspmaxitp(rspmaxitpidx(end):end) = rspmaxitp(rspmaxitpidx(end));

rspminitp(1:rspminitpidx(1)) = rspminitp(rspminitpidx(1));
75 rspminitp(rspminitpidx(end):end) = rspminitp(rspminitpidx(end));

% subtracting the mins from the maxs gives a set of amplitudes
rspamp = rspmaxitp - rspminitp;
uvals = [find(diff(rspamp)~=0); length(rspamp)];
80 rspampu = rspamp(uvals);

if bDisplayPfAmp,
    figPfAmp = figure;

```

```

subplot(211),
85 plot(t, rspamp, 'b', ...
      t(uvals), rspampu, 'bo', ...
      t, rspmaxitp, 'r:', ...
      t, rspminitp, 'g:',...
      t, rspout, 'k', ...
90      t(rspmax(:,1)), rspmax(:,2), 'rs', ...
      t(rspmin(:,1)), rspmin(:,2), 'go')
xlabel('Time (hr)')
ylabel('N.<FL> (Arb)');

95 subplot(212),
hist(rspampu(~isnan(rspampu)))
xlabel('Amplitude (Arb)')
ylabel('Counts')

100 subplot(211),
title(sprintf('%s Period (Frequency):%5.2fhr (%5.4fhr^-1)', ...
             stExpt.id, stExpt.period, 1/stExpt.period), ...
      'interpreter', 'none');

105 set(figPfAmp, 'name', sprintf('PF AMP: %5.3f (%s)', ...
                               stExpt.period, stExpt.id));
if bPrint,
    drawnow;
    print(figPfAmp);
110 end;
if bSave,
    drawnow;
    exportfig(figPfAmp, [stExpt.figname, '__pf_amp'], cFormats);
end;
115 end;

% Way of determining phase-shift recommended by Mike Ferry:
% Loop through the response peaks and find the 'latest' signal peak
% that occurred before the response peak.  measure the time difference
120 % as the phase-shift

shift = [];
sigpktimes = t(sigmax(:, 1));
for iRspPk = 1:size(rspmax,1),
125     rsppktime = t(rspmax(iRspPk, 1));
     sigpktime = sigpktimes(find(sigpktimes <= rsppktime, 1, 'last'));

     shift = [shift; rsppktime - sigpktime];

130 % theres a chance that there isn't a correspondding signal peak for
% the response peak.  this will produce an empty shift, but since
% values are just appended to the end of the array, this shouldn't
% have an effect on the results.

135 end;

% repeat for the rsp/sig minima
sigpktimes = t(sigmin(:, 1));
for iRspPk = 1:size(rspmin,1),
140     rsppktime = t(rspmin(iRspPk, 1));

```

```

sigpktime = sigpktimes(find(sigpktimes <= rsppktime, 1, 'last'));

shift = [shift; rsppktime - sigpktime];

145 end;

if bDisplayPfPhi,
    figPfPhi = figure;
    x = normrange(sigout);
150    y = normrange(rspout);

    % plot the overlay of the renormalized signal and response
    subplot(311)
    plot(    t, x, 'r', ...
155          t(sigmax(:,1)), x(sigmax(:,1)), 'g.', ...
          t(sigmin(:,1)), x(sigmin(:,1)), 'g*', ...
          t, y, 'b', ...
          t(rspmax(:,1)), y(rspmax(:,1)), 'rs', ...
160          t(rspmin(:,1)), y(rspmin(:,1)), 'rd' ...
          );
    xlabel('Time (hr)');
    ylabel('');

    % plot the shift trace
165    subplot(312)
    plot(shift, 'b');
    xlabel('Index');
    ylabel('Value (hr)');

170    % plot the histogram of shifts
    subplot(313)
    hist(shift);
    xlabel('Value (hr)');
    ylabel('Counts');

175    subplot(311)
    title(sprintf('%s Period (Frequency):%5.2fhr (%5.4fhr^-1)', ...
        stExpt.id, stExpt.period, 1/stExpt.period), ...
        'interpreter', 'none');

180    %set(figPfPhi, 'windowstyle', 'docked');
    set(figPfPhi, 'name', sprintf('PF PHI: %5.3f (%s)', ...
        stExpt.period, stExpt.id));

185    if bPrint,
        drawnow;
        print(figPfPhi);
    end;
    if bSave,
190        drawnow;
        exportfig(figPfPhi, [stExpt.figname, '__pf_phi'], cFormats);
    end;
end;

195 phipf = shift*1/stExpt.period;
    amppf = rspampu(~isnan(rspampu));

```

```

function hFigs = ttmc_analysis_summarize(sStrain, frq, amp, amperr, ...
                                     phi, phierr, varargin);
% generates the summary plots of phase and amplitude data for the ttmc
% frequency scan study. requires at least three vectors: frq, amp, and
5 % phi.

opts = getopt('varargin');
bVerbose = parseopts('verbose', opts, false);

10 cOutput = parseopts('output', opts, {});
    if iscell(cOutput) && ~isempty(cOutput),
        bPrint = ~isempty(strmatch('print', lower(cOutput), 'exact'));
        bSave = ~isempty(strmatch('save', lower(cOutput), 'exact'));
    else
15     bPrint = false;
        bSave = false;
    end;

    if bSave,
20     % only process the formats option if 'save' is specified. no
        % initialization is ok to do since everything is behind the bSave
        % boolean
        cFormats = parseopts('formats', opts, {'fig', 'png'});
        if ~iscell(cFormats) || isempty(cFormats),
25         % just in case users specify something weird resort to defaults
            cFormats = {'fig', 'png'};
        end;
    end;

30 % assume everything is sorted as it should be

    % =====
    % plot the amplitude data
    figFrqVsAmp = figure;
35 if ~isempty(amperr),
        errorbar(frq, amp, amperr, 'bo-')
    else
        plot(frq, amp, 'bo-')
    end;
40 title('Amplitude Ratio vs. Frequency');
    xlabel('Frequency (rads/hr)');
    ylabel('Amplitude Ratio (Arb.)');
    set(figFrqVsAmp, 'name', 'Amplitude Ratio vs. Frequency')
    if bPrint,
45     drawnow;
        print(figFrqVsAmp);
    end;
    if bSave,
        drawnow;
50     exportfig(figFrqVsAmp, [sStrain '_summary__freq_vs_amp'], cFormats);
    end;

    figPerVsAmp = figure;
    if ~isempty(amperr),
55     errorbar(2*pi./frq, amp, amperr, 'bo-')
    else

```

```

        plot(2*pi./frq, amp, 'bo-')
    end;
    title('Amplitude Ratio vs. Period');
60 xlabel('Period (hr)');
    ylabel('Amplitude Ratio (Arb.)');
    set(figPerVsAmp, 'name', 'Amplitude Ratio vs. Period')
    if bPrint,
        drawnow;
65     print(figPerVsAmp);
    end;
    if bSave,
        drawnow;
        exportfig(figPerVsAmp, [sStrain '_summary__per_vs_amp'], cFormats);
70 end;

% =====
% plot the phase data
figFrqVsPhi = figure;
75 if ~isempty(phierr),
    errorbar(frq, phi, phierr, 'bo-')
    else
        plot(frq, phi, 'bo-')
    end;
80 title('Phase Shift vs. Frequency');
    xlabel('Frequency (rads/hr)');
    ylabel('Phase Shift (Rads)');
    set(figFrqVsPhi, 'name', 'Phase Shift vs. Frequency')
    if bPrint,
85     drawnow;
        print(figFrqVsPhi);
    end;
    if bSave,
        drawnow;
90     exportfig(figFrqVsPhi, [sStrain '_summary__freq_vs_phi'], cFormats);
    end;

    figPerVsPhi = figure;
    if ~isempty(phierr),
95     errorbar(2*pi./frq, phi, phierr, 'bo-')
    else
        plot(2*pi./frq, phi, 'bo-')
    end;
    title('Phase Shift vs. Period');
100 xlabel('Period (hr)');
    ylabel('Phase Shift (Rads)');
    set(figPerVsPhi, 'name', 'Phase Shift vs. Period')
    if bPrint,
        drawnow;
105     print(figPerVsPhi);
    end;
    if bSave,
        drawnow;
        exportfig(figPerVsPhi, [sStrain '_summary__per_vs_phi'], cFormats);
110 end;

hFigs = [figFrqVsAmp, figPerVsAmp, figFrqVsPhi, figPerVsPhi];

```

ttmc.expts.m


```

% blob data analysis of frequency runs
% runs with 2,0.2,0.25pct r,g,d:
%      period imival  dir          clr  strain    cfpch  yfpch
%      (hrs)   (min)
5 expts = {
    6.00    5.0    'g:/09132006/03', 'b', 'yph499', 2, -1;
    4.50    5.0    'g:/09302006/02', 'y', 'yph499', 2, -1;
    3.00    5.0    'g:/09092006/03', 'r', 'yph499', 2, -1;
    2.25    5.0    'g:/09302006/03', 'm', 'yph499', 2, -1;
10    1.50    5.0    'g:/09132006/01', 'g', 'yph499', 2, -1;
    1.125   5.0    'g:/10052006/01', 'c', 'yph499', 2, -1;
    0.75    2.5    'g:/09132006/02', 'k', 'yph499', 2, -1;
    6.00    5.0    'g:/10082006/01', 'b', 'k699', 3, 2;
    4.50    5.0    'g:/10082006/02', 'b', 'k699', 3, 2;
15    3.00    5.0    'g:/09262006/05', 'b', 'k699', 3, 2;
    2.25    5.0    'g:/10082006/03', 'b', 'k699', 3, 2;
    1.50    5.0    'g:/09262006/04', 'b', 'k699', 3, 2;
    1.125   5.0    'g:/10082006/04', 'b', 'k699', 3, 2;
    0.75    5.0    'g:/10152006/01', 'b', 'k699', 3, 2;
20    };

```

E.2 Model simulation code

```

----- ttmc_gengludata.m -----
% generate glucose trace from experiment data
curdir = pwd;

exptdirs = { ...
5     './experiment\cfp\k699';
     './experiment\cfp\yph499';
};

for j = 1:length(exptdirs),
10    cd(exptdirs{j});
    csFiles = dir('*__raw.fig');
    csFiles = {csFiles(:).name}';
    csFiles = flipud(csFiles);

15    cd(curdir);
    for i = 1:length(csFiles),
        nTokenLoc = strmatch('_', csFiles{i}');
        sExptID = csFiles{i}(1:nTokenLoc(2)-1);
        [x y] = getfigdata([exptdirs{j} filesep csFiles{i}], ...
20        313, {'color', 'b'});

        save(sExptID, 'x', 'y');
    end
end
-----

----- yph499_trial_runner2.m -----
if ~exist('bIsFunction', 'var'),
    clear;
    nPeriod = 6;          % hours
    tdoub=8;             % in hours

```

```

5
    tStart = clock;
    sSimID = sprintf('TD%.2f_P%.3f', tdoub, nPeriod);
    fprintf('%s : ', sSimID);
    sSimID = strrep(sSimID, '.', 'p');
10
    end;

    mpath = mfilename('fullpath');
    mpath = fileparts(mpath);
15
    glufile = [mpath, filesep, 'gludata', filesep, ...
              strrep(sprintf('yph499_%.3f', nPeriod), '.', 'p'), '.mat'];
    fprintf('loading: %s\n', glufile);

20 load(glufile);
    trialt=x;
    trialglu=y;

    dt=mean(diff(trialt))*60;
25 nInterval = dt;
    period=nPeriod*60;
    tmax=max(trialt)*60;
    ttrans=10000;
    galtime=0.001;
30 tdouble=tdoub;
    ge=11;  %( 11=0.2% gal)
    glue=0;
    glucosemax=.25*11/.2;
    w=2*pi/period;
35
    trialt=trialt*60;
    trialglu=trialglu/(max(trialglu));

    gtf=9;
40 galtf=1.5;

    b=2;
    eps=.23*galtf;
    a=0.001*galtf;
45 s=.75*galtf;
    g=0.16*galtf;
    d=log(2)/(tdouble*60)*galtf;
    ktr=4350;
    kmtr=1;  %/molectomM;
50 atr=30;  %1;
    galx=8;
    a2=0.001*gtf;
    g2=.16*gtf;
    eps2=.4*gtf;
55 s2=.75*gtf;
    d2=gtf*log(2)/(tdouble*60);
    ktr2=4350;
    kmtr2=1;
    atr2=30;
60 rc=6;
    q=1.6;

```

```

b2=1.2;
glux=14;

65 x0=0;
m0=0;
gi0=0;
x20=0;
m20=0;
70 glui0=0;

ics=[m0 x0 gi0 m20 x20 glui0];
abstol=(1e-6)*ones(1,length(ics));
options = odeset('RelTol',1e-6,'AbsTol',abstol);
75

fprintf('Simulating Steady State : ');

ge=0;
80 glue=0;
gA=0;
tspan=[0 ttrans];
params=[a g b eps s d ktr kmtr atr ge galx a2 g2 b2 ...
        eps2 s2 d2 ktr2 kmtr2 atr2 glue rc q glux gA w];
85 [T,Y]=ode15s(@(t,y) simp_gg_trial_ode2(t,y,params,trialt,trialglu),...
               tspan, ics, options);

tSS = clock;
fprintf('%6.3f : ',etime(tSS, tStart));
90 fprintf('Simulating Perturbation Run : ');

dd=size(Y);
ics=Y(dd(1),:);
ge=11;
95 gA=0;
params=[a g b eps s d ktr kmtr atr ge galx a2 g2 b2 ...
        eps2 s2 d2 ktr2 kmtr2 atr2 glue rc q glux gA w];
tspan=[0 galtime];
[T,Y]=ode15s(@(t,y) simp_gg_trial_ode2(t,y,params,trialt,trialglu),...
100            tspan, ics, options);

dd=size(Y);
ics=Y(dd(1),:);
105 ge=11;
gA=glucosemax;
params=[a g b eps s d ktr kmtr atr ge galx a2 g2 b2 ...
        eps2 s2 d2 ktr2 kmtr2 atr2 glue rc q glux gA w];
tspan=[0:dt:tmax];
110 [T2,Y2]=ode15s(@(t,y) simp_gg_trial_ode2(t,y,params,trialt,trialglu),...
                 tspan, ics, options);

tPR = clock;
fprintf('%6.3f : ',etime(tPR, tSS));
115 fprintf('%6.3f ;\n',etime(tPR, tStart));

T=T2;
Y=Y2;

```

```

gallout=Y(:,2);
120
lmin=0;
lmax=55; % 6hr=65; %4.5hr=55;
mmin=min(gallout);
mmax=max(gallout);
125 % sf=(lmax-lmin)/(mmax-mmin);
sf=lmax/mmax;

gallout=gallout*sf;
%gallout=gallout-gallout(1)+lmin;
130
gal2out=gallout;

% hold on
% plot(T/60,gallout,'k')
135
gluout = trialglu;
% gluout=(gA/2*(1+sin(w*T)));
% plot(T/60,gA-gluout,'k')

```

```

_____ k699_trial_runner2.m _____
if ~exist('bIsFunction', 'var'),
    clear;
    nPeriod = 6;          % hours
    tdoub=8;             % in hours
5
    tStart = clock;
    sSimID = sprintf('TD%.2f_P%.3f', tdoub, nPeriod);
    fprintf('%s : ',sSimID);
    sSimID = strrep(sSimID, '.', 'p');
10
end;

mpath = mfilename('fullpath');
mpath = fileparts(mpath);
15
glufile = [mpath, filesep, 'gludata', filesep, ...
          strrep(sprintf('k699_%.3f', nPeriod), '.', 'p'), '.mat'];
fprintf('loading: %s\n', glufile);

20 load(glufile);
    trialt=x;
    trialglu=y;

    dt=mean(diff(trialt))*60;
25 nInterval = dt;
    period=nPeriod*60;
    tmax=max(trialt)*60;
    ttrans=10000;
    galtime=0.001;
30 tdouble=tdoub;
    ge=11; % ( 11=0.2% gal)
    glue=0;
    glucosemax=.25*11/.2;
    w=2*pi/period;
35

```

```

trialt=trialt*60;
trialglu=trialglu/(max(trialglu));

gtf=80;
40 galtf=1;

b=1.5;
eps=.23*galtf;
a=0.001*galtf;
45 s=.75*galtf;
g=0.16*galtf;
d=log(2)/(tdouble*60)*galtf;
ktr=4350;
kmtr=1; %/molectomM;
50 atr=30; %1;
galx=.6;
a2=0.001*gtf;
g2=.16*gtf;
eps2=.4*gtf;
55 s2=.75*gtf;
d2=gtf*log(2)/(tdouble*60);
ktr2=4350;
kmtr2=1;
atr2=30;
60 rc=6;
q=1.6;
b2=1.2;
glux=72;

65 x0=0;
m0=0;
gi0=0;
x20=0;
m20=0;
70 glui0=0;

ics=[m0 x0 gi0 m20 x20 glui0];
abstol=(1e-6)*ones(1,length(ics));
options = odeset('RelTol',1e-6,'AbsTol',abstol);
75

fprintf('Simulating Steady State : ');

ge=0;
80 glue=0;
gA=0;
tspan=[0 ttrans];
params=[a g b eps s d ktr kmtr atr ge galx a2 g2 b2 ...
        eps2 s2 d2 ktr2 kmtr2 atr2 glue rc q glux gA w];
85 [T,Y]=ode15s(@(t,y) simp_gg_trial_ode(t,y,params,trialt,trialglu),...
               tspan, ics, options);

tSS = clock;
fprintf('%6.3f : ',etime(tSS, tStart));
90 fprintf('Simulating Perturbation Run : ');

dd=size(Y);

```

```

    ics=Y(dd(1),:);
    ge=11;
95 gA=0;
    params=[a g b eps s d ktr kmtr atr ge galx a2 g2 b2 ...
            eps2 s2 d2 ktr2 kmtr2 atr2 glue rc q glux gA w];
    tspan=[0 galtime];
    [T,Y]=ode15s(@(t,y) simp_gg_trial_ode(t,y,params,trialt,trialglu),...
100         tspan, ics, options);

    dd=size(Y);
    ics=Y(dd(1),:);
105 ge=11;
    gA=glucosemax;
    params=[a g b eps s d ktr kmtr atr ge galx a2 g2 b2 ...
            eps2 s2 d2 ktr2 kmtr2 atr2 glue rc q glux gA w];
    tspan=[0:dt:tmax];
110 [T2,Y2]=ode15s(@(t,y) simp_gg_trial_ode(t,y,params,trialt,trialglu),...
        tspan, ics, options);

    tPR = clock;
    fprintf('%6.3f : ',etime(tPR, tSS));
115 fprintf('%6.3f ;\n',etime(tPR, tStart));

    T=T2;
    Y=Y2;
    gallout=Y(:,2);
120
    lmin=0;
    lmax=400; % 6hr=65; %50.34;
    mmin=min(gallout);
    mmax=max(gallout);
125 % sf=(lmax-lmin)/(mmax-mmin);
    sf=lmax/mmax;

    gallout=gallout*sf;
    %gallout=gallout-gallout(1)+lmin;
130
    gal2out=gallout;

    % hold on
    % plot(T/60,gallout,'g')
135
    gluout = trialglu;
    % gluout=(gA/2*(1+sin(w*T)));
    % plot(T/60,gA-gluout,'g')

```

```

_____ simp_gg_trial_ode2.m _____
function dy=simp_glu_gal_ode(t,y,params,ttime,tglu)
    dy=zeros(6,1);

    a=params(1);
5    g=params(2);
    b=params(3);
    eps=params(4);
    s=params(5);
    d=params(6);

```

```

10  ktr=params(7);
    kmtr=params(8);
    atr=params(9);
    ge=params(10);
    galx=params(11);
15  a2=params(12);
    g2=params(13);
    b2=params(14);
    eps2=params(15);
    s2=params(16);
20  d2=params(17);
    ktr2=params(18);
    kmtr2=params(19);
    atr2=params(20);
    glue=params(21);
25  rc=params(22);
    q=params(23);
    glux=params(24);
    gA=params(25);
    w=params(26);

30  m=y(1);
    x=y(2);
    gi=y(3);
    m2=y(4);
35  x2=y(5);
    glui=y(6);

    [ttt ind]=max(t<ttime);
    glut=tglu(ind);

40  %glue=gA/2*(1+sin(w*t));
    glue=gA*glut;

    dy(1)=(a+eps*gi^b)/(galx^b+gi^b)*(rc^q/(rc^q+x2^q))-g*m;
45  dy(2)=s*m-d*x;
    dy(3)=ktr*x*((ge-gi)/(kmtr+ge+gi+atr/kmtr*ge*gi));

    dy(4)=(a2+eps2*glui^b2)/(glux^b+glui^b2)-g2*m2;
    dy(5)=s2*m2-d2*x2;
50  dy(6)=ktr*x*((glue-glui)/(kmtr+glue+glui+atr/kmtr*glue*glui)) ...
        +ktr2*x2*((glue-glui)/(kmtr2+glue+glui+atr2/kmtr2*glue*glui));

```

```

_____ getfigdata.m _____
function [x y] = getfigdata(sFigFileName, iSubplotID, cSearchSpec)

    if ischar(sFigFileName),
        open(sFigFileName)
5  elseif ishandle(sFigFileName),
        figure(sFigFileName)
    end;

    subplot(iSubplotID);
10  xy = get( ...
        findobj( ...

```

```
                get(gca, 'children'), ...
                'type', 'line', '-and', cSearchSpec ...
            ), ...
15         {'xdata', 'ydata'} ...
            );
    if ischar(sFigFileName),
        close(gcf);
    end;
20     x = xy{1};
        y = xy{2};
    end
```

Bibliography

- [1] Murat Acar, Attila Becskei, and Alexander van Oudenaarden. Enhancement of cellular memory by reducing stochastic transitions. *Nature*, 435(7039):228–232, May 2005.
- [2] J. R. Anderson, D. T. Chiu, R. J. Jackman, O. Cherniavskaya, J. C. McDonald, H. K. Wu, S. H. Whitesides, and G. M. Whitesides. Fabrication of topologically complex three-dimensional microfluidic systems in pdms by rapid prototyping. *Analytical Chemistry*, 72(14):3158–3164, 2000.
- [3] Jr. Austriaco, N. R. Review: to bud until death: the genetics of ageing in the yeast, *Saccharomyces*. *Yeast*, 12(7):623–30, Jun 1996.
- [4] F. K. Balagadde, L. You, C. L. Hansen, F. H. Arnold, and S. R. Quake. Long-term monitoring of bacteria undergoing programmed population control in a microchemostat. *Science*, 309(5731):137–40, 2005.
- [5] F. K. Balagadde, L. C. You, F. H. Arnold, and S. R. Quake. Programmed population control by cell-cell communication in microfluidic chemostats. *Biophysical Journal*, 88(1):519a–519a, 2005.
- [6] B. Banerjee, S. Balasubramanian, G. Ananthkrishna, T. V. Ramakrishnan, and G. V. Shivashankar. Tracking operator state fluctuations in gene expression in single cells. *Biophysical Journal*, 86(5):3052–3059, 2004.
- [7] C. K. Barlowe and D. R. Appling. Molecular genetic analysis of *saccharomyces cerevisiae* c1-tetrahydrofolate synthase mutants reveals a noncatalytic function of the *ade3* gene product and an additional folate-dependent enzyme. *Mol Cell Biol*, 10(11):5679–5687, Nov 1990.
- [8] A. Becskei, B. Séraphin, and L. Serrano. Positive feedback in eukaryotic gene networks: cell differentiation by graded to binary response conversion. *EMBO J.*, 20(10):2528–35, May 2001.
- [9] A. Becskei and L. Serrano. Engineering stability in gene networks by autoregulation. *Nature*, 405(6786):590–3, Jun 2000.
- [10] David J Beebe, Glennys A Mensing, and Glenn M Walker. Physics and applications of microfluidics in biology. *Annu Rev Biomed Eng*, 4:261–286, 2002.

- [11] P. J. Bhat, D. Oh, and J. E. Hopper. Analysis of the GAL3 signal transduction pathway activating GAL4 protein-dependent transcription in *Saccharomyces cerevisiae*. *Genetics*, 125(2):281–291, Jun 1990.
- [12] R. Byron Bird, Warren E Stewart, and Edwin N Lightfoot. *Transport phenomena*. John Wiley & Sons, New York, NY, 1st edition, 1960.
- [13] W. J. Blake, M. Kærn, C. R. Cantor, and J. J. Collins. Noise in eukaryotic gene expression. *Nature*, 422(6932):633–7, Apr 2003.
- [14] F. R. Blattner, G. Plunkett, C. A. Bloch, N. T. Perna, V. Burland, M. Riley, J. Collado-Vides, J. D. Glasner, C. K. Rode, G. F. Mayhew, J. Gregor, N. W. Davis, H. A. Kirkpatrick, M. A. Goeden, D. J. Rose, B. Mau, and Y. Shao. The complete genome sequence of *Escherichia coli* K-12. *Science*, 277(5331):1453–1474, Sep 1997.
- [15] Erez Braun and Naama Brenner. Transient responses and adaptation to steady state in a eukaryotic gene regulation system. *Phys Biol*, 1(1-2):67–76, Jun 2004.
- [16] Long Cai, Nir Friedman, and X. Sunney Xie. Stochastic protein expression in individual cells at the single molecule level. *Nature*, 440(7082):358–362, Mar 2006.
- [17] Robert E Campbell, Oded Tour, Amy E Palmer, Paul A Steinbach, Geoffrey S Baird, David A Zacharias, and Roger Y Tsien. A monomeric red fluorescent protein. *Proc Natl Acad Sci U S A*, 99(12):7877–7882, Jun 2002.
- [18] R. Y. Chein and S. H. Tsai. Microfluidic flow switching design using volume of fluid model. *Biomedical Microdevices*, 6(1):81–90, 2004.
- [19] B. G. Chung, L. A. Flanagan, S. W. Rhee, P. H. Schwartz, A. P. Lee, E. S. Monuki, and N. L. Jeon. Human neural stem cell growth and differentiation in a gradient-generating microfluidic device. *Lab on a Chip*, 5(4):401–406, 2005.
- [20] R. A. Clayton, O. White, K. A. Ketchum, and J. C. Venter. The first genome from the third domain of life. *Nature*, 387(6632):459–462, May 1997.
- [21] Scott Cookson, Natalie Ostroff, Wyming Lee Pang, Dmitri Volfson, and Jeff Hasty. Monitoring dynamics of single-cell gene expression over multiple cell cycles. *Molecular Systems Biology*, 1(1):msb4100032–E1–msb4100032–E6, 2005.
- [22] B. P. Cormack, R. H. Valdivia, and S. Falkow. Facs-optimized mutants of the green fluorescent protein (gfp). *Gene*, 173(1 Spec No):33–38, 1996.
- [23] S. E. Cowan, D. Liepmann, and J. D. Keasling. Development of engineered biofilms on poly-l-lysine patterned surfaces. *Biotechnology Letters*, 23(15):1235–1241, 2001.
- [24] J. C. Crocker and D. G. Grier. Methods of digital video microscopy for colloidal studies. *Journal of Colloid and Interface Science*, 179(1):298–310, 1996.
- [25] Culbertson, Ramsey, and Ramsey. Electroosmotically induced hydraulic pumping on microchips: differential ion transport. *Anal Chem*, 72(10):2285–2291, May 2000.

- [26] Ozlem Demir and Isil Aksan Kurnaz. An integrated model of glucose and galactose metabolism regulated by the gal genetic switch. *Computational Biology and Chemistry*, 30(3):179–192, 2006.
- [27] Stephen K. W. Dertinger, Daniel T. Chiu, Noo Li Jeon, and George M. Whitesides. Generation of gradients having complex shapes using microfluidic networks. *Anal. Chem.*, 73(6):1240 – 1246, 2001.
- [28] Wen-Bin Du, Qun Fang, Qiao-Hong He, and Zhao-Lun Fang. High-throughput nanoliter sample introduction microfluidic chip-based flow injection analysis system with gravity-driven flows. *Anal Chem*, 77(5):1330–1337, Mar 2005.
- [29] D. C. Duffy, O. J. A. Schueller, S. T. Brittain, and G. M. Whitesides. Rapid prototyping of microfluidic switches in poly(dimethyl siloxane) and their actuation by electro-osmotic flow. *Journal of Micromechanics and Microengineering*, 9(3):211–217, 1999.
- [30] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.
- [31] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.
- [32] D. Erickson, D. Sinton, and D. Q. Li. Joule heating and heat transfer in poly(dimethylsiloxane) microfluidic systems. *Lab on a Chip*, 3(3):141–149, 2003.
- [33] David W Galbraith, Rangasamy Elumalai, and Fang Cheng Gong. Integrative flow cytometric and microarray approaches for use in transcriptional profiling. *Methods Mol Biol*, 263:259–280, 2004.
- [34] Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403:339 – 342, 2000.
- [35] A. Gedvilaite and K. Sasnauskas. Control of the expression of the *ade2* gene of the yeast *saccharomyces cerevisiae*. *Curr Genet*, 25(6):475–479, Jun 1994.
- [36] Alex Groisman, Caroline Lobo, HoJung Cho, J. Kyle Campbell, Yann S Dufour, Ann M Stevens, and Andre Levchenko. A microfluidic chemostat for experiments with bacterial and yeast cells. *Nat Methods*, 2(9):685–689, Sep 2005.
- [37] C. C. Guet, M. B. Elowitz, W. H. Hsing, and S. Leibler. Combinatorial synthesis of genetic networks. *Science*, 296(5572):1466–1470, 2002.
- [38] Nicholas J Guido, Xiao Wang, David Adalsteinsson, David McMillen, Jeff Hasty, Charles R Cantor, Timothy C Elston, and J. J. Collins. A bottom-up approach to gene regulation. *Nature*, 439(7078):856–860, Feb 2006.
- [39] Richard Haberman. *Elementary applied partial differential equations: with Fourier series and boundary value problems*. Prentice Hall, Upper Saddle River, NJ, 3rd edition, 1998.

- [40] Carl L Hansen, Morten O A Sommer, and Stephen R Quake. Systematic investigation of protein phase behavior with a microfluidic formulator. *Proc Natl Acad Sci U S A*, 101(40):14431–14436, Oct 2004.
- [41] J. Hasty, D. McMillen, and J. J. Collins. Engineered gene circuits. *Nature*, 420(6912):224–30, Nov 2002.
- [42] J. Hasty, D. McMillen, F. Isaacs, and J. J. Collins. Computational studies of gene regulatory networks: *in numero* molecular biology. *Nat. Rev. Genet.*, 2(4):268–79, Apr 2001.
- [43] Kristy M Hawkins and Christina D Smolke. The regulatory roles of the galactose permease and kinase in the induction response of the GAL network in *Saccharomyces cerevisiae*. *J Biol Chem*, 281(19):13485–13492, May 2006.
- [44] My Hedhammar, Maria Stenvall, Rosa Lnnborg, Olof Nord, Olle Sjlin, Hjalmar Brismar, Mathias Uhln, Jenny Ottosson, and Sophia Hober. A novel flow cytometry-based method for analysis of expression levels in *escherichia coli*, giving information about precipitated and soluble protein. *J Biotechnol*, 119(2):133–146, Sep 2005.
- [45] C. C. Hong, J. W. Choi, and C. H. Ahn. A novel in-plane passive microfluidic mixer with modified tesla structures. *Lab on a Chip*, 4(2):109–113, 2004.
- [46] J. W. Hong, V. Studer, G. Hang, W. F. Anderson, and S. R. Quake. A nanoliter-scale nucleic acid processor with parallel architecture. *Nature Biotechnology*, 22(4):435–439, 2004.
- [47] T. Ideker, V. Thorsson, J. A. Ranish, R. Christmas, J. Buhler, J. K. Eng, R. Bumgarner, D. R. Goodlett, R. Aebersold, and L. Hood. Integrated genomic and proteomic analyses of a systematically perturbed metabolic network. *Science*, 292(5518):929–934, May 2001.
- [48] Daniel Irimia, Su-Yang Liu, William G Tharp, Azadeh Samadani, Mehmet Toner, and Mark C Poznansky. Microfluidic system for measuring neutrophil migratory responses to fast switches of chemical gradients. *Lab Chip*, 6(2):191–198, Feb 2006.
- [49] Farren J. Isaacs, Jeff Hasty, Charles R. Cantor, and J. J. Collins. Prediction and measurement of an autoregulatory genetic module. *PNAS*, 100(13):7714–7719, 2003.
- [50] R. F. Ismagilov, D. Rosmarin, P. J. A. Kenis, D. T. Chiu, W. Zhang, H. A. Stone, and G. M. Whitesides. Pressure-driven laminar flow in tangential microchannels: an elastomeric microfluidic switch. *Analytical Chemistry*, 73(19):4682–4687, 2001.
- [51] H. Ito, Y. Fukuda, K. Murata, and A. Kimura. Transformation of intact yeast cells treated with alkali cations. *J Bacteriol*, 153(1):163–168, Jan 1983.
- [52] X. Y. Jiang, S. Takayama, P. LeDuc, D. E. Ingber, and G. M. Whitesides. Studying mammalian cell’s reaction to topographical features fabricated by an unconventional method. *Biophysical Journal*, 82(1):164a–164a, 2002.

- [53] X. Y. Jiang, S. Takayama, X. P. Qian, E. Ostuni, H. K. Wu, N. Bowden, P. LeDuc, D. E. Ingber, and G. M. Whitesides. Controlling mammalian cell spreading and cytoskeletal arrangement with conveniently fabricated continuous wavy features on poly(dimethylsiloxane). *Langmuir*, 18(8):3273–3280, 2002.
- [54] M. Johnston and J-H. Kim. Glucose as a hormone: receptor-mediated glucose sensing in the yeast *Saccharomyces cerevisiae*. *Biochem Soc Trans*, 33(Pt 1):247–252, Feb 2005.
- [55] Aneta Kaniak, Zhixiong Xue, Daniel Macool, Jeong-Ho Kim, and Mark Johnston. Regulatory network connecting two glucose signal transduction pathways in *Saccharomyces cerevisiae*. *Eukaryot Cell*, 3(1):221–231, Feb 2004.
- [56] M. Kasahara, E. Shimoda, and M. Maeda. Amino acid residues responsible for galactose recognition in yeast gal2 transporter. *J Biol Chem*, 272(27):16721–16724, Jul 1997.
- [57] Seon-Young Kim and YongSung Kim. Genome-wide prediction of transcriptional regulatory elements of human promoters using gene expression and promoter analysis data. *BMC Bioinformatics*, 7:330, 2006.
- [58] H. Kobayashi, M. Kaern, M. Araki, K. Chung, T. S. Gardner, C. R. Cantor, and J. J. Collins. Programmable cells: Interfacing natural and engineered gene networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(22):8414–8419, 2004.
- [59] T. Fettah Kosar, Chihchen Chen, Nicholas L. Stucky, and Albert Folch. Arrays of microfluidically-addressable nanoholes. *Journal of Biomedical Nanotechnology*, 1(2):161–167, June 2005.
- [60] Karsten Kruse and Frank Julicher. Oscillations in cell biology. *Current Opinion in Cell Biology*, 17(1):20–26, 2005.
- [61] Edo Kussell, Roy Kishony, Nathalie Q Balaban, and Stanislas Leibler. Bacterial persistence: a model of survival in changing environments. *Genetics*, 169(4):1807–1814, Apr 2005.
- [62] Edo Kussell and Stanislas Leibler. Phenotypic diversity, population growth, and information in fluctuating environments. *Science*, 309(5743):2075–2078, Sep 2005.
- [63] N. C. Kyrpides. Genomes online database (gold 1.0): a monitor of complete and ongoing genome projects world-wide. *Bioinformatics*, 15(9):773–774, Sep 1999.
- [64] G. B. Lee, C. I. Hung, B. J. Ke, G. R. Huang, and B. H. Hwei. Micromachined pre-focused 1 x n flow switches for continuous sample injection. *Journal of Micromechanics and Microengineering*, 11(5):567–573, 2001.
- [65] G. B. Lee, B. H. Hwei, and G. R. Huang. Micromachined pre-focused m x n flow switches for continuous multi-sample injection. *Journal of Micromechanics and Microengineering*, 11(6):654–661, 2001.

- [66] J. Li, S. Wang, W. J. VanDusen, L. D. Schultz, H. A. George, W. K. Herber, H. J. Chae, W. E. Bentley, and G. Rao. Green fluorescent protein in *Saccharomyces cerevisiae*: real-time studies of the GAL1 promoter. *Biotechnol Bioeng*, 70(2):187–196, Oct 2000.
- [67] M. E. Lidstrom and D. R. Meldrum. Life-on-a-chip. *Nature Reviews Microbiology*, 1(2):158–164, 2003.
- [68] F. Lin, W. Saadi, S. W. Rhee, S. J. Wang, S. Mittal, and N. L. Jeon. Generation of dynamic temporal and spatial concentration gradients using microfluidic devices. *Lab on a Chip*, 4(3):164–167, 2004.
- [69] O. Lipan and W. H. Wong. The use of oscillatory signals in the study of genetic networks. *Proc Natl Acad Sci U S A*, 102(20):7063–8, 2005.
- [70] M. P. Longhese, R. Frascini, P. Plevani, and G. Lucchini. Yeast *pip3/mec3* mutants fail to delay entry into s phase and to slow dna replication in response to dna damage, and they define a functional link between *mec3* and dna primase. *Mol Cell Biol*, 16(7):3235–3244, Jul 1996.
- [71] H. Lu, L.Y. Koo, W.M. Wang, D.A. Lauffenburger, L.G. Griffith, and K.F. Jensen. Microfluidic shear devices for quantitative analysis of cell adhesion. *Anal. Chem.*, 76(18):5257–5264, 2004.
- [72] Yong Luo, Dapeng Wu, Shaojiang Zeng, Hongwei Gai, Zhicheng Long, Zheng Shen, Zhongpeng Dai, Jianhua Qin, and Bingcheng Lin. Double-cross hydrostatic pressure sample injection for chip ce: variable sample plug volume and minimum number of electrodes. *Anal Chem*, 78(17):6074–6080, Sep 2006.
- [73] Andreas Maier, Bernhard Vlker, Eckhard Boles, and Gnter Fred Fuhrmann. Characterisation of glucose transport in *saccharomyces cerevisiae* with plasma membrane vesicles (countertransport) and intact cells (initial uptake) with single *hxt1*, *hxt2*, *hxt3*, *hxt4*, *hxt6*, *hxt7* or *gal2* transporters. *FEMS Yeast Res*, 2(4):539–550, Dec 2002.
- [74] M. A. McMurray and D. E. Gottschling. Aging and genetic instability in yeast. *Curr. Opin. Microbiol.*, 7(6):673–9, Dec 2004.
- [75] Jerome T Mettetal, Dale Muzzey, Juan M Pedraza, Ertugrul M Ozbudak, and Alexander van Oudenaarden. Predicting stochastic gene expression dynamics in single cells. *Proc Natl Acad Sci U S A*, 103(19):7304–7309, May 2006.
- [76] Patrick Morier, Christine Vollet, Philippe E Michel, Frdric Reymond, and Jol S Rossier. Gravity-induced convective flow in microfluidic systems: electrochemical characterization and application to enzyme-linked immunosorbent assay tests. *Electrophoresis*, 25(21-22):3761–3768, Nov 2004.
- [77] R. K. Mortimer and J. R. Johnston. Genealogy of principal strains of the yeast genetic stock center. *Genetics*, 113(1):35–43, May 1986.

- [78] C. Neils, Z. Tyree, B. Finlayson, and A. Folch. Combinatorial mixing of microfluidic streams. *Lab on a Chip*, 4(4):342–350, 2004.
- [79] E. M. Ozbudak, M. Thattai, H. N. Lim, B. I. Shraiman, and A. van Oudenaarden. Multistability in the lactose utilization network of escherichia coli. *Nature*, 427(6976):737–740, 2004.
- [80] S. Ozcan and M. Johnston. Three different regulatory mechanisms enable yeast hexose transporter (HXT) genes to be induced by different levels of glucose. *Mol Cell Biol*, 15(3):1564–1572, Mar 1995.
- [81] S. Ozcan, T. Leong, and M. Johnston. Rgt1p of *Saccharomyces cerevisiae*, a key regulator of glucose-induced genes, is both an activator and a repressor of transcription. *Mol Cell Biol*, 16(11):6419–6426, Nov 1996.
- [82] Juan M Pedraza and Alexander van Oudenaarden. Noise propagation in gene networks. *Science*, 307(5717):1965–1969, Mar 2005.
- [83] D. Porro and F. Srienc. Tracking of individual cell cohorts in asynchronous *saccharomyces-cerevisiae* populations. *Biotechnology Progress*, 11(3):342–347, 1995.
- [84] M. Ptashne. *A Genetic Switch: Phage λ and Higher Organisms*. Cell Press: Blackwell Scientific Publications, Cambridge, MA, 2nd edition, 1992.
- [85] R. Qiao and N. R. Aluru. A compact model for electroosmotic flows in microfluidic devices. *Journal of Micromechanics and Microengineering*, 12(5):625–635, 2002.
- [86] S. Quake. Biological large scale integration. *Abstracts of Papers of the American Chemical Society*, 227:U116–U116, 2004.
- [87] J. Ramos, K. Szkutnicka, and V. P. Cirillo. Characteristics of galactose transport in *Saccharomyces cerevisiae* cells and reconstituted lipid vesicles. *J Bacteriol*, 171(6):3539–3544, Jun 1989.
- [88] C. V. Rao, D. M. Wolf, and A. P. Arkin. Control, exploitation and tolerance of intracellular noise. *Nature*, 420(6912):231–7, Nov 2002.
- [89] S. W. Rhee, A. M. Taylor, C. H. Tu, D. H. Cribbs, C. W. Cotman, and N. L. Jeon. Patterned cell culture inside microfluidic devices. *Lab Chip*, 5(1):102–7, 2005.
- [90] M. Ronen and D. Botstein. Transcriptional response of steady-state yeast cultures to transient perturbations in carbon source. *Proc Natl Acad Sci U S A*, 103(2):389–94, 2006.
- [91] V. Saadi, S. J. Wang, F. Lin, M. C. Nguyen, and N. L. Jeon. Application of microfluidic chambers for cancer cell migration: the effects of egf gradient patterns on cell movement. *Faseb Journal*, 17(5):A1369–A1369, 2003.
- [92] S. Schlautmann, G. A. J. Besselink, R. Prabhu, and R. B. M. Schasfoort. Fabrication of a microfluidic chip by uv bonding at room temperature for integration of temperature-sensitive layers. *Journal of Micromechanics and Microengineering*, 13(4):S81–S84, 2003.

- [93] Mark A Sheff and Kurt S Thorn. Optimized cassettes for fluorescent protein tagging in *Saccharomyces cerevisiae*. *Yeast*, 21(8):661–670, Jun 2004.
- [94] L. N. Sierkstra, N. P. Nouwen, J. M. Verbakel, and C. T. Verrips. Analysis of glucose repression in *Saccharomyces cerevisiae* by pulsing glucose to a galactose-limited continuous culture. *Yeast*, 8(12):1077–1087, Dec 1992.
- [95] R. S. Sikorski and P. Hieter. A system of shuttle vectors and yeast host strains designed for efficient manipulation of DNA in *Saccharomyces cerevisiae*. *Genetics*, 122(1):19–27, May 1989.
- [96] Abraham D. Stroock, Stephan K. W. Dertinger, Armand Ajdari, Igor Mezic, Howard A. Stone, and George M. Whitesides. Chaotic mixer for microchannels. *Science*, 295(5555):647–651, 2002.
- [97] N. Tesla. U. S. Patent No. 1,329,559, 03 Feb 1920.
- [98] M. Thattai and B. I. Shraiman. Metabolic switching in the sugar phosphotransferase system of *Escherichia coli*. *Biophysical Journal*, 85(2):744–754, 2003.
- [99] M. Thattai and A. van Oudenaarden. Attenuation of noise in ultrasensitive signaling cascades. *Biophysical Journal*, 82(6):2943–2950, 2002.
- [100] D. M. Thompson, K. R. King, K. J. Wieder, M. Toner, M. L. Yarmush, and A. Jayaraman. Dynamic gene expression profiling using a microfabricated living cell array. *Analytical Chemistry*, 76(14):4098–4103, 2004.
- [101] T. Thorsen, S. J. Maerkl, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.
- [102] A. Tourovskaia, X. Figueroa-Masot, and A. Folch. Differentiation-on-a-chip: A microfluidic platform for long-term cell culture studies. *Lab on a Chip*, 5(1):14–19, 2005.
- [103] R. Y. Tsien. The green fluorescent protein. *Annu Rev Biochem*, 67:509–544, 1998.
- [104] T. L. Ulery, D. A. Mangus, and J. A. Jaehning. The yeast *imp1* gene is allelic to *gal2*. *Mol Gen Genet*, 230(1-2):129–135, Nov 1991.
- [105] M. A. Unger, H. P. Chou, T. Thorsen, A. Scherer, and S. R. Quake. Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science*, 288(5463):113–116, 2000.
- [106] Mahesh Uttamchandani, Jun Wang, and Shao Q Yao. Protein and small molecule microarrays: powerful tools for high-throughput proteomics. *Mol Biosyst*, 2(1):58–68, Jan 2006.
- [107] Daniele Venturoli and Bengt Rippe. Ficoll and dextran vs. globular proteins as probes for testing glomerular permselectivity: effects of molecular size, shape, charge, and deformability. *Am J Physiol Renal Physiol*, 288(4):F605–613, 2005.
- [108] Malkhey Verma, Paiké J Bhat, and K. V. Venkatesh. Steady-state analysis of glucose repression reveals hierarchical expression of proteins under *Mig1p* control in *Saccharomyces cerevisiae*. *Biochem J*, 388(Pt 3):843–849, Jun 2005.

- [109] Dmitri Volfson, Jennifer Marciniak, William J Blake, Natalie Ostroff, Lev S Tsimring, and Jeff Hasty. Origins of extrinsic variability in eukaryotic gene expression. *Nature*, 439(7078):861–864, Feb 2006.
- [110] Glenn M Walker and David J Beebe. A passive pumping method for microfluidic devices. *Lab Chip*, 2(3):131–134, Aug 2002.
- [111] John R Walker and Tim Wiltshire. Databases of free expression. *Mamm Genome*, 17(12):1141–1146, Dec 2006.
- [112] G. M. Whitesides, J. Jiang, S. Sia, V. Linder, B. Parviz, A. Sigel, and J. Lee. Soft lithography and bioanalysis. *Abstracts of Papers of the American Chemical Society*, 227:U113–U113, 2004.
- [113] G. M. Whitesides, E. Ostuni, S. Takayama, X. Y. Jiang, and D. E. Ingber. Soft lithography in biology and biochemistry. *Annual Review of Biomedical Engineering*, 3:335–373, 2001.
- [114] Jiejun Wu, Laura T Smith, Christoph Plass, and Tim H-M Huang. Chip-chip comes of age for genome-wide functional analysis. *Cancer Res*, 66(14):6899–6902, Jul 2006.
- [115] Z. Y. Wu, N. Xanthopoulos, F. Reymond, J. S. Rossier, and H. H. Girault. Polymer microchips bonded by o-2-plasma activation. *Electrophoresis*, 23(5):782–790, 2002.
- [116] Younan Xia and George M. Whitesides. Soft lithography. *Angewandte Chemie International Edition*, 37(5):550 – 575, 1998.
- [117] W. Xiong and J. E. Ferrell. A positive-feedback-based bistable 'memory module' that governs a cell fate decision. *Nature*, 426(6965):460–465, 2003.
- [118] M. Y. Ye, Q. Fang, X. F. Yin, and Z. L. Fang. Studies on bonding techniques for poly (dimethylsiloxane) microfluidic chips. *Chemical Journal of Chinese Universities-Chinese*, 23(12):2243–2246, 2002.
- [119] Kwang-Seok Yun and Euisik Yoon. Micro/nanofluidic device for single-cell-based assay. *Biomedical Microdevices*, 7(1):35–40, 2005.
- [120] FK Zimmerman and KD Entian. *Yeast Sugar Metabolism*. Technomic Publishing Company, Inc, Lancaster, Pennsylvania, 1st edition, 1997.
- [121] Martin Zimmermann, Heinz Schmid, Patrick Hunziker, and Emmanuel Delamarque. Capillary pumps for autonomous capillary systems. *Lab Chip*, 7(1):119–125, Jan 2007.