Designing from End-to-End and Learning Control Policies on the Edge with Data-driven Optimization: Applications to Adaptive Plasma Medicine

By

Kimberly J. Chan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Chemical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Spring 2024

Designing from End-to-End and Learning Control Policies on the Edge with Data-driven Optimization: Applications to Adaptive Plasma Medicine

Abstract

Designing from End-to-End and Learning Control Policies on the Edge with Data-driven Optimization: Applications to Adaptive Plasma Medicine

by

Kimberly J. Chan

Doctor of Philosophy in Chemical Engineering

University of California, Berkeley

Associate Professor Ali Mesbah, Chair

Cold atmospheric plasmas (CAPs) are becoming a breakthrough technology in a variety of materials processing and characterization applications, including in plasma medicine. CAP jets (CAPJs) are a versatile tool in plasma medicine because they can be a low-cost, portable, point-of-care solution for a variety of biomedical applications. However, selecting operational parameters of CAPJs (or CAPs in general) remains an open challenge due to a variety of factors, including variability in patients (i.e., target interface), variability in CAPJ operation, sensitivity to disturbances and environmental conditions, and difficult-to-model dynamics of CAPs resulting in uncertain predictions about CAP-interface interactions. Predictive control has become the state-of-the-art in addressing aspects of the safety, reproducibility, and efficacy of CAP treatments. This dissertation addresses two open aspects of CAP control, specifically designing feasible embedded control systems for point-of-care CAPJs and designing individualized CAP treatment regimens. Together, these two aspects represent the overarching objective of this dissertation: enabling point-of-care devices for precision plasma medicine.

CAPJs for biomedical applications are often touted for their portability and point-of-care use. Additionally, medicine as a field is moving towards more targeted approaches to patient healthcare due to the influx of data from personal devices (e.g., smart wearables) that track health trends and physical activity and due to the importance of considering diverse patient profiles for equitable and efficacious medical treatments. This trend (part of a tendency towards "edge computing") combined with the nonlinear, multi-variable CAP dynamics calls for embedded control policies that are capable of implementation on resource-limited hardware. The first part of this dissertation provides a novel fusion of hardware and software design (aka "hardware-software co-design") of control policies to find optimal and feasible embedded control policies on resource-limited hardware. In particular, key elements of the end-to-end design pipeline include the digital control policy, the physical computing

hardware, and the closed-loop performance measures of interest such as chemical/biological effects of CAPs on target interfaces. We demonstrate that a data-driven optimization framework based on Bayesian optimization (BO), which can simultaneously incorporate the control policy design and hardware considerations when implementing the control policy, can effectively design feasible embedded control policies that target multiple objectives. An estimation of the Pareto frontier (i.e., trade-off curve) can be generated via hardware-in-the-loop simulations and used to inform the design of real-time control policies.

Several applications in plasma medicine require repeated treatments to realize therapeutically effective treatment outcomes to avoid overdosing and/or to treat long-term conditions. Prior works have illustrated predictive control strategies are capable of safely delivering CAP treatments to patients, but these strategies generally rely on underlying assumptions of individual subject characteristics (i.e., empirical models based on population data). This consideration necessitates adaptive treatments that are updated via observations of treatment outcomes, which can be addressed through data-driven optimization. In simulations and experiments, we demonstrated that deep learning-based control policies, which are amenable to resource-limited hardware, can be updated directly using multi-objective BO. We demonstrated how deep learning-based control policies can be updated to find the optimal trade-offs in treatment objectives when characteristics of individual subjects may differ from the population. In a complementary direction, we developed a novel strategy to safely explore the individualized objective space without compromising on performance improvements. We demonstrated that our safe explorative BO strategy finds a balance between overly-cautious exploration that may get stuck at local optima and overly-eager exploration that may violate safety-critical constraints.

The primary focus of this dissertation was on the therapeutic benefits of CAPs. The final contribution of this dissertation investigated a novel aspect of CAPs for biomedical use: (biological) material characterization. We demonstrated that CAPs are uniquely capable of producing minimally destructive effects during interactions with biological tissues that can be used to identify and classify different tissue types. A key aspect of this finding is that real-time chemical and electrical measurements of plasma-tissue interactions can be analyzed in physics-informed ways and fed into machine learning strategies to predict the type of a biological tissue. Results from this study can have significant implications in non-invasive early skin cancer detection systems and/or in real-time surgical assistance.

To conclude, this dissertation presented results that illustrate an end-to-end journey from the design of physical computing hardware to the design of digital control policies to the design and characterization of (bio)chemical outcomes of plasma treatments in medicine. This dissertation established that data-driven optimization is a versatile tool to regulate and personalize the outcomes of CAP treatments. For medicine, BO mimics the doctor-patient interaction, and thus provides a natural augmentation to the medical toolkit. Future work may involve addressing additional challenges regarding connected devices and data-driven strategies (i.e., (cyber)security, privacy, distributed deployment), fusion of physics-structured learning with data, and evaluation of such methods in preclinical and clinical studies. The findings in this dissertation were grounded in plasma medicine, but can be broadly applicable to other non-equilibrium plasma applications, e.g., semiconductor processing.

To my parents and my partner, whose support has kept me strong throughout graduate school.

Contents

List of Figures

List of Tables

Acknowledgments

This PhD has been a journey that has challenged me every step of the way. Being the first in my family to go to graduate school meant that I knew next-to-nothing to start, but came out with a wealth of knowledge and experiences at the end. All of it could not have been done without an incredible support system.

First, I would like to thank my advisor, Prof. Ali Mesbah, for being the "end-to-end" reason for my success throughout the PhD. At the beginning, he took a chance and allowed me to join his lab, where we embarked on an ambitious journey to combine some of the most complex fields, plasma science, control theory, machine learning, and biology/medicine. Throughout the PhD, he has challenged me, guided me, and given me many opportunities to flourish as a researcher. From submitting to conferences and fellowships to interning at NASA to facilitating collaborations across the world, I have had a great time exploring the limits of what I could do as a graduate student, and Ali has been the one to provide the support and advocacy whenever I needed it. I will be forever grateful of our time working together during my PhD.

Next, I'd like to thank my various collaborators who have helped me learn much faster than anything I could've done on my own. Prof. Joel Paulson was instrumental in helping me work out any kinks of our work, providing a fresh perspective and critically challenging us when we were stuck in the weeds. Joel's energy is also contagious as his passion towards his and our work would always revitalize my efforts after our conversations. Dr. Augusto Stancampiano and the whole team at GREMI in Orléans, France were a tremendous plasma(-biology) collaboration team. Namely, Augusto provided a warm welcome and guided me throughout my stay in France, and Sébastien Dozias helped me to retrofit their "Plasma Gun" when my electrical experience was lacking. Our continued collaboration helped (and hopefully will continue to help) our Berkeley group expand into plasma(-biology) experimental work. Prof. Javad Mohammadpour Velni and Dr. Yajie Bao helped provide an alternative perspective on our control problems by tackling the modeling side. Additionally, side projects with Dr. Kapil Sawlani and Ging Martin at Lam Research were interesting and gave me a view on the semiconductor processing side of plasma processes. I am also thankful for the opportunity to (briefly) work with and meet Prof. Leili Afsah-hejri (and a quick shoutout to the Wenjun Zhang lab for providing the space to do the biology experiments). While our interactions were brief, she taught us how to prepare and conduct culturing experiments, which I hope to pick up again sometime in the future.

Aside from pure research, I'd like to thank the entire Mesbah lab, past and present. First and foremost, I am immensely grateful for Victor Miller and Ketong Shao, who took the leap of faith alongside me to join the Mesbah lab in Fall 2019. They have not only been the best lab-mates, but also lifelong friends; they've been there with me through the ups and downs, and we've challenged and learned so much from each other through lengthy whiteboard discussions and long nights in the lab. Next, the older graduate students, whose work provided the foundation of my work, Dr. Dogan Gidon and Dr. Angelo Bonzanini. Though I didn't personally meet Dogan, my thesis work would not have been possible without his thesis and the control-oriented plasma jet that he was instrumental in creating. (By extension, big thanks go out to members of the Graves lab that helped him build it.) Angelo's work in control was also foundational to my work, as it helped me build the major components of our intermediate advanced control layer. His code provided examples of and sped up my understanding of robust formulations of model predictive control. He and Ketong also helped me learn and troubleshoot the plasma jet when I was first learning to use it. I am also thankful to Dr. Georgios Makrygiorgios for his idea and work for a collaboration with data-driven optimization and policy search, which is now a section of this dissertation and opened the door for our personalized medicine take on adaptive plasma treatments. I'm grateful to have worked with Dr. Diogo Rodrigues, who let me be part of his work for my first journal paper. I'm thankful for all of the undergraduates that I mentored, but most of all Kelci Skinner, who was a talented, detailed-oriented mentee throughout her entire time with our group. I'm thankful for Dr. Jared O'leary, who provided never-ending energy and supportive advice when things didn't go to plan. I'm grateful for all of our younger graduate students (former and present), Mira Khare, who helped build out more of our lab culture by hosting potlucks, Shoubhanik Nath, Joshua Ip, Melanie Huynh, and Thomas (Tommy) Banker for having engaging conversations about research and the department, and I am grateful for Dr. Nima Moini for his advice on surviving the final parts of the PhD.

I am also thankful to have met several visiting graduate students and/or scholars throughout my time in the Mesbah lab. I thank Prof. Alessandro do Nascimento Vargas for being our temporary in-house electronics/electrical engineering expert, when, again, my skills were lacking, and I thank Prof. Rosileide (Rose) de Oliveira Lopes for being my desk-mate and having conversations about research and non-research topics. I am grateful to have met Thuc (Thuci) Nguyen and Kiet Hoang who brought their brilliance over from Germany and Norway, respectively, and provided engaging discussions on controls research. I am grateful to have met Dr. Kwanghyun Cho who provided some industrial insights where our work could be useful and participated in our group's first-ever secret Santa event despite being unable to make it in person.

Outside of the immediate research circle, there have been several people at the Department of Chemical and Biomolecular Engineering and UC Berkeley that have helped me tremendously throughout my PhD. I am thankful to the professors in my qualifying and thesis committees: Prof. Karthik Shekhar, Prof. Francesco Borrelli, Prof. Nitash Balsara, and Prof. Markita Landry who each sat through two qualifying exams to provide me invaluable feedback and advice on my research. In particular, Karthik was the one (aside from Ali) to lift me up out of my first qualifying exam and has been supportive of me throughout the rest of the PhD. I am thankful to all of the staff, including Carlet Altamirano for doing most of the behind-the-scenes work at various events and fixing any and all of the bureaucratic hurdles that graduate students have to jump through, Joseph Nolan for ensuring that I got paid on time or fixing incorrect/late pay and for ensuring that I got my reimbursements in a timely fashion, Esayas Kelkile, Alexei Anderson and the rest of the building management team for helping our lab fix/figure out pressure imbalance problems, fume hood issues, strange smells, etc.

I am grateful for having had the opportunity to work as an intern at NASA (something that had never even crossed my mind). Dr. William (Jeremy) Coupe was my technical mentor who kept us interns thinking about the latest advances in artificial intelligence and machine learning and led us through projects related to air traffic management when that was not our area of expertise. Shannon Zelinski and Jaewoo Jung were each my supervisor at different points; they helped me chart out project milestones and provided advice and help in my own professional development. Rene Mai, Annie Lee, and Kathleen Gee were the other interns under Jeremy who provided engaging discussions and interesting results on tailoring language models for aviation-specific use.

From more of a social perspective, I am grateful for the supportive student body within our department. Namely, the cohort of 2019 who've been a diverse group of students to talk to and hang out with. In particular, the "chiccy crew" was a small sub-group that enjoyed (Asian) food experiences (Justin Bui, Andrew Ying, Erika Ding) and a small games group that played various Dungeons and Dragons (DnD) or DnD-like campaigns and/or various video and board games (Grace Anderson, Victor, Erika, and on occasion Ketong and Ahmad Alkadri). To GSAC, I am grateful to all of the people past and present who volunteered their time to create and organize the graduate student body, namely those that took on the difficult roles of president and DEI chair(s). Their work has fostered community and changes that have provided (incrementally) better experiences for the graduate students within the department.

My biggest thanks goes out to my family and my partner, Xinyang Chen, who have given me unwavering love, support, and encouragement throughout my PhD and, more importantly, life. My parents have, of course, always been there for me since I was born, but graduate school was uncharted territory for us. It was truly an eye-opening and heartwarming experience seeing them be happy and proud of my accomplishments, but also having their words of encouragement push me through tough times. My dad has always been the one to nudge me to try difficult and challenging things, and my mom showed me how to be the best that I can be. My brother and my extended family have also been motivating factors throughout my PhD. Xinyang has been the most incredible partner of this PhD journey. From the start, he cleared any reservations that I had about moving to Berkeley and starting a long-distance relationship. Since then, he's celebrated every win with me, picked me up through any setbacks, and pushed me to new limits (e.g., starting an MBA at Haas and making me go to more networking events than I care for).

My time at Berkeley has been a unique, and at times testing, experience that has helped me grow into not only a better researcher, but also a better person overall. There were numerous other people and small interactions that I didn't list here that have really summed together to make my PhD what it is now. I am grateful for what I have accomplished and am excited for the future ahead of me!

Chapter 1

Introduction

This chapter motivates and introduces the necessary background to explore the frontier of predictive control on the edge for cold atmospheric plasma in medicine. This background includes a description of the transition from segregated design to end-to-end design, a brief review of plasma medicine and control for plasma medicine, and an introduction to the framework that this dissertation uses to address challenges in control on the edge for plasma medicine.

1.1 Background and Motivation

The Fourth Industrial Revolution marks the (current) time period where digital technology is woven into the fabric of daily human life [9,10]. This period has been vaguely defined by the emergence of technological breakthroughs grounded in digital systems, e.g., cyberphysical systems (CPS), internet of things (IoT), artificial intelligence (AI), etc. One important aspect of these innovations is the hardware components that make up portable devices and the control programs that lie within them. This dissertation explores strategies to bridge the gaps in computing on the edge and advanced control for nonlinear, constrained, multivariable, and/or uncertain systems. Namely, advances in the realm of machine learning (ML) and optimal control have enabled a form of learning-based predictive control to update control policies in a data-driven manner [11–13]. Extensive exploration of the fourth state of matter, plasma, has created a new field of research, plasma medicine, with the goal to exploit plasma's complex chemical, electrical, thermal properties in innovative biomedical applications [14–17]. In plasma medicine, beneficial impacts of learning-based predictive control include a mathematically-backed strategy to identify new treatment protocols, the ability to personalize medicine, and the ability to bring updated predictive control to the point-of-use [18, 19]. This dissertation proposes that learning-based predictive control is crucial to address the needs of adaptive and individualized plasma medicine at the edge.

The following sections of this chapter will provide the necessary background to explore the frontier of control on the edge for cold plasma devices in medicine. First, we elaborate on

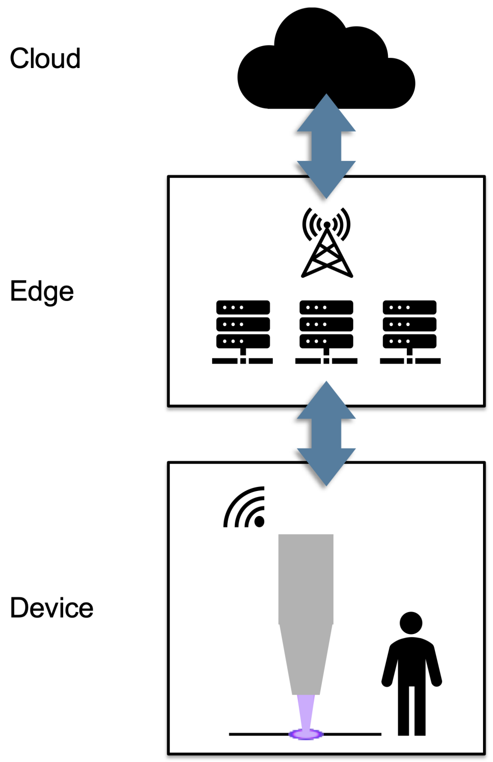


Figure 1.1: Edge computing relies on communication between the cloud, edge servers, and end devices to decentralize the computational needs of individual devices. Future (plasma) medical technologies will depend on the edge computing architecture as shown to enable adaptive and individualized treatments and treatment regimens.

the unique challenges that stem from the end-to-end design of control at the edge. Next, we provide a brief overview of cold atmospheric plasma's potential in medicine and motivate the key challenge to making its future viable: optimal control. Then, we describe the central idea of this dissertation, which involves casting the control of a plasma system as an optimization problem that is solved in a data-driven manner. Finally, we discuss the contributions and organization of this dissertation.

1.2 End-to-end Control Design: From System Specifications to Hardware Design to Software Design

Scientific computing entails the use of advanced computational methods to solve scientific and engineering problems, and edge computing is becoming more relevant with the emergence of many CPS and IoT devices. The core idea of edge computing lies in bringing as much computation as possible to "the edge," or near the endpoint, where the data or end-user is located, as illustrated in Figure 1.1. The shift towards edge computing is necessary as more and more devices are connected wirelessly and the need for computational resources to support those devices grow exponentially. This means that individual devices at the user interface must increasingly take on computational tasks themselves. Advanced control and automation are leading contributors to the push towards edge computing. Smart devices and autonomous vehicles are examples of the ubiquitous computing that are available to users in the modern era; these devices rely on distributed sensing and scientific computations to interact with the complex environments they are in. However, the need to take on more scientific computations can often be at odds with the low computational capabilities of the hardware at the edge. As such, a new paradigm of design is essential to enabling advanced technologies at the end-user level.

Medicine is one field in which edge computing is hoping to gain traction [20–23]. The promise of personalized and adaptive treatments bodes well for patients, as individualized treatments lead to better overall health outcomes [24–26]. However, designing (edge) devices that exist at the human-health interface involves a set of challenges revolving around safety, reproducibility, efficacy, and privacy [21]. In doing so, it is imperative to make decisions based on the full pipeline of design. As illustrated in Figure 1.2, this includes considerations of the chemical/biological, computing hardware, and digital sides, which make the design process inherently interwoven and practically difficult to model and challenging to interpret. Advances have been made individually in each category: on the digital side, complex control policies, such as model predictive control (MPC) [27–29], can handle a variety of multi-input, multi-output nonlinear and complex systems; on the computing hardware side, graphics processing units (GPUs) [30], field programmable gate arrays (FPGAs) [31, 32] and others have enabled faster real-time computations; and on the chemical/biological side, where studies show progress in characterizing the complex interactions of plasma and (bio)interfaces using real-time available measurements [33, 34].

This dissertation aims to address parts of each dimension towards individualized and adaptive plasma medicine, and while medicine remains the focus of this dissertation, elements of this design paradigm can be applied various other applications involving plasma interactions.

1.3 Application to Cold Atmospheric Plasmas in Medicine

Plasmas are the "fourth fundamental state of matter" characterized by its composition of neutral species, radicals, ions, and electrons. Non-equilibrium plasmas are weakly ionized gases whose mean kinetic energy of the electrons (T_e) is much greater than that of the heavy particles (T_g, i.e., $T_e \gg T_g$) [35, 36]. In particular, cold atmospheric plasmas (CAPs)

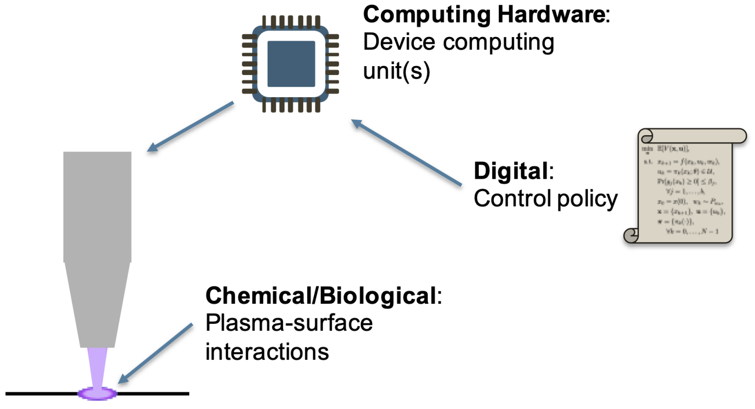


Figure 1.2: Design elements of a (plasma) medical device at the edge. For devices at the edge, the full pipeline of design should include information about the chemical and biological outcomes, considerations about the physical computing hardware, and considerations regarding the digital control policy (i.e., the embedded software).

are a non-equilibrium plasma that exists at atmospheric pressure conditions [37]. CAPs are uniquely capable of locally generating reactive chemical species, ions, electric fields, photons, and thermal effects, which can be delivered to heat- and pressure-sensitive targets [36, 37]. Plasma medicine is an innovative field at the intersection of plasma physics and clinical medicine and has inspired a significant increase in the research studies on their effectiveness as an alternative or complementary therapy [14–17]. Plasma medicine established its roots in the 1990s and continues to advance clinically. Table 1.1 lists a few recent advances in the plasma medicine research community.

Despite the latest progress, an open challenge in plasma medicine still lies in determining the proper conditions in which to use and operate plasma, i.e., designing treatment protocols based on real-time observations of plasma characteristics and treatment efficacy. In plasma medicine, the main form of control problem lies in what is called the "dose delivery" problem, wherein some variable amount of plasma effects (be it thermal, chemical, electrical, or radiative) are delivered to an interface. While this can be a simple problem when handling chemical medications, a plasma dose is poorly defined due to the multivariable plasma-interface interactions [58, 59]. Further, operational conditions of plasma devices (e.g., translation across an interface, interface/subject variability) pose additional challenges to overcome when applying plasma treatments [14, 60]. Efforts to address the dose delivery problem have been investigated with promising results [1, 2, 61–65]. The challenges highlighted in these works that continue to play a major role in adaptation of plasma treatment protocols in this work include:

1. CAP systems are difficult to model in that physical models have high computational complexity (presents a challenge in embedded, real-time control) and/or are incomplete

Category	**Contributions**	**Reference(s)**
Disinfection and Sterilization	Commercial sterilization system for pharmaceuticals	[38]
	Hand sanitation	[39], [40]
	Pathogenic molecule (e.g., bacteria, viruses, prions) inactivation	[41], [42], [43], [44], [45]
Wound Healing	Treatment of chronic venous ulcers	[46]
	Post-surgery treatment of infected/disturbed wounds (acute wound healing)	[47], [48]
	Combination with conventional antiseptics	[49]
Plasma Oncology	Selective cancer cell targeting	[50], [51]
	Assistance in inducing immunogenic cell death	[52], [53]
	Surgical aide	[54], [55]
Safety	CAP effects on proteins	[56]
	Minimally-destructive against ocular tissues	[57]

Table 1.1: A non-exhaustive list of recent advances in plasma medicine.

descriptions of the physical system (presents a challenge in accurate control, which has consequences in terms of efficacy and safety).

2. CAPs and biomedical systems are sensitive to environmental factors and individual subjects. CAPs have run-to-run variations even under nearly-identical operating conditions [60] and exhibit steep gradients in temperature and reactive species concentrations [58]. Furthermore, the nature of biological systems results in stochastic outcomes, and the lack of diversity in historical medical studies poses a need establish patient-centric healthcare [66].

3. Real-time and non-interfering sensing of CAP treatment efficacy is limited. CAPs and its effects on biological materials require characterization tools that are high-speed and expensive, whereas quantifying and evaluating the effect of CAPs on biological systems occurs over a much longer timescale, which has posed a challenge in understanding the relation between CAP operation and biological outcomes.

Prior work focused on the algorithmic complexities of control policies that are necessary to operate CAPs under the challenges listed above. Instead, this dissertation aims to explore

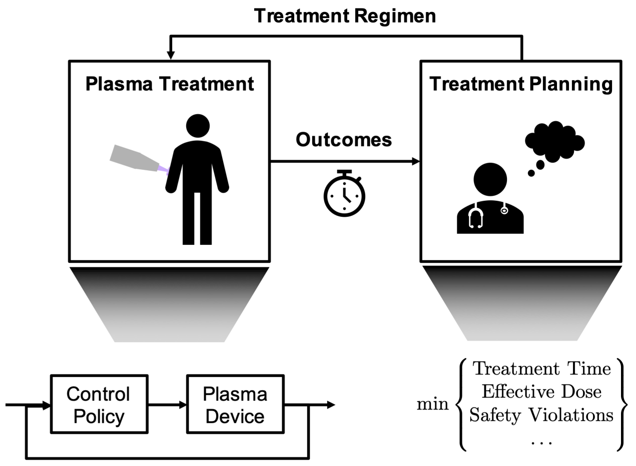


Figure 1.3: Illustration of the doctor-patient interaction to determine and optimize an overall treatment regimen. In plasma medicine, there exists an inner loop of feedback control to ensure reliable and reproducible plasma treatments [1–3]. Then, over some timescale separation, physicians and practitioners must take observations of the biochemical outcomes of the plasma treatment to inform future decisions regarding additional treatment(s) in an outer optimization loop. The focus of this dissertation is on the outer optimization loop.

an iterative framework that adds a layer of optimization-based treatment adaptation on the biological or therapeutic outcomes.

1.4 Data-driven Optimization for Adaptation of Predictive Control Policies

For multi-treatment protocols in general medicine, it is typical for patients and physicians to have regular check-ups to adjust/modify the treatment regimen as necessary. This can proceed as illustrated in Figure 1.3. For plasma medicine, prior works have shown that plasma treatment must incorporate some form of advanced feedback control policy to ensure reliable and reproducible plasma treatments [1–3]. This feedback control policy must be on the order of millisecond to second time scales due to the fast dynamics of ion and electron interactions and plasma-interface chemistry, while the biological effects are determined minutes to hours to days after the plasma application [34]. This dissertation proposes a data-driven means to supplement or inform a physician's strategy for plasma treatment regimens due to the additional layer of feedback control and the lack of understanding in plasma-biological interactions and outcomes.

Consider an optimization problem in the context of plasma medicine: a constrained minimization problem in which the objective(s) and constraint(s) are black-box due to the lack of knowledge about the plasma-interface interactions

$$\min_{\theta} \left\{ \{J_i(\theta)\}_{i=1}^m \quad \text{subject to } \{c_j(\theta)\}_{j=1}^p \geq 0 \right\}, \tag{1.1}$$

where $\theta \in \Theta$ is a set of adjustable parameters of a plasma treatment, whether that be plasma operating parameters or control policy parameters, $\{J_i : \Theta \to \mathbb{R}\}_{i=1}^m$ are a set of m objectives/treatment outcomes, and $\{c_j : \Theta \to \mathbb{R}\}_{j=1}^p$ are a set of p safety-critical constraints that describe whether a trajectory/plasma treatment is safe (≥ 0) or unsafe (< 0).The optimization problem (1.1) cannot be solved by standard gradient-based optimization methods (e.g., gradient descent) due to the following features of J_i (and c_j):

1. The mathematical structure of J_i may not be known in closed-form, especially if J_i is a result of a plasma treatment under an optimization-based control policy (i.e., one that may involve the implicit solution of another optimization problem, e.g., model predictive control).

2. The dynamics of a plasma treatment and the effects of CAPs on biological interfaces (aka "dose") are not known exactly; thus the gradients of J_i cannot be computed.

To address these challenges, derivative-free optimization (DFO) is one avenue of exploration since DFO methods are general and make little-to-no assumptions regarding the objective(s) and constraint(s) [67,68]. However, most DFO methods still require many evaluations on the true system (i.e., plasma treatments) [69]. Bayesian optimization (BO) is a class of DFO methods that were specifically designed to tackle noisy and expensive-to-evaluate objective(s) [70]. BO recasts the optimization problem (1.1) as a sequential learning problem, similar to the doctor-patient interaction to evaluate the efficacy and safety of a treatment regimen. The BO algorithm consists of two main components: i) a predictive probabilistic surrogate model representing the objective(s) and constraint(s) and ii) an "acquisition function" that quantifies the benefit of querying the posterior objective(s) and constraint(s). Further details about BO will be given in the ensuing chapters; here, we highlight a few key ways that BO can be used to address open questions in plasma medicine as investigated in this dissertation. BO can be used to

1. modify parameters θ of arbitrary control policies, be it MPC or deep learning policies that are amenable to embedded implementations for edge devices,

2. modify arbitrary design parameters of the embedded control or hardware implementation, meaning that the digital aspect can be designed in conjunction with the computing hardware aspect of a plasma-biology system,

3. solve a multi-objective and/or constrained problem, which is common to medical practice when there are various considerations involving efficacy of treatment, patient comfort, patient safety, and/or device capability, and

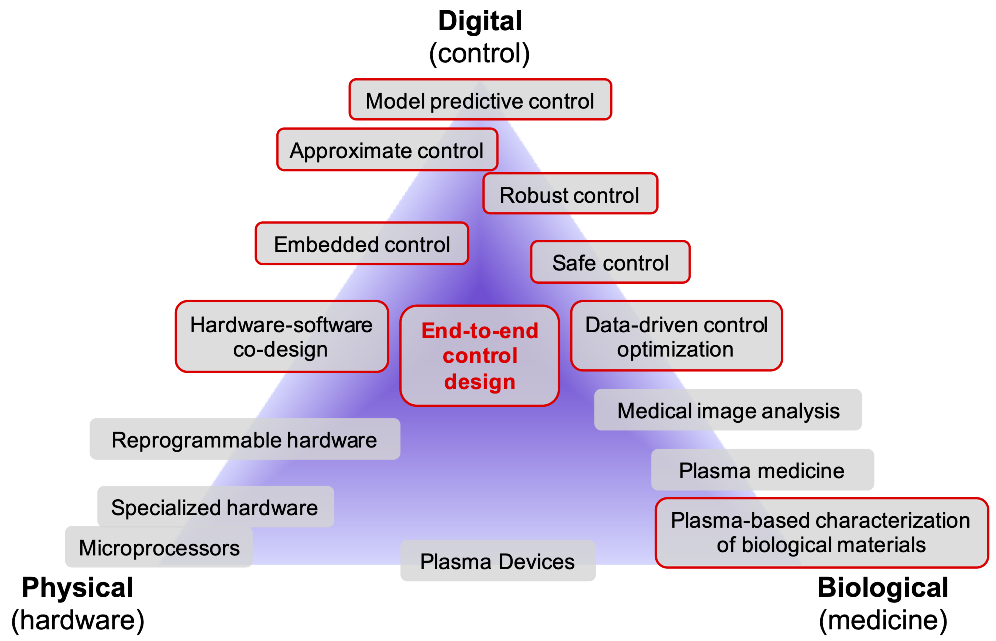


Figure 1.4: Concept diagram of the interplay between digital (embedded control policy), physical (computing hardware of a device), and biological (treatment outcomes) design for cold atmospheric plasmas in medicine. Areas explored in this dissertation are outlined in red.

4. adapt control policies in a safe, explorative manner, where safety is guaranteed with high probability without loss in performance gains.

Additionally, preference learning BO has recently been investigated for personalized plasma medicine and demonstrates how patient feedback can be incorporated into a data-driven framework to more effectively guide the control policy search based on user-centric preferences [71].

1.5 Contributions

The overarching objective of this dissertation is to **investigate the end-to-end design of embedded control systems on point-of-care devices that enable individualized plasma treatment regimens in plasma medicine**. To tackle this overarching goal, we use BO as a framework to inform treatment regimens through (i) optimal co-design of computing hardware and digital control policy for embedded control and (ii) safe and multi-objective adaptation of plasma treatment regimens for optimized and individualized control over timescale-separated biochemical outcomes, and (iii) we investigate advancements in the characterization of plasma effects on biological materials. Figure 1.4 provides a conceptual diagram to illustrate the concepts and methods investigated in this thesis. Aim (i) involves the fusion of embedded control and hardware-software co-design, and when necessary, replace complex control (such as robust formulations) with approximate control. Aim (ii) connects aspects of the application (e.g., safety-critical constraints, embedded control architectures) with control policy adaptation and highlights how data-driven control optimization can be used to efficiently explore plasma operating conditions that lead to desired biochemical outcomes. Finally, Aim (iii) presents a novel perspective to evaluate biological materials using CAPs, which poses a unique opportunity to use the same CAP device (under different operational modes) for not only medical therapies but also for diagnostics. Table 1.2 summarizes the contributions that led to the culmination of this thesis. Works highlighted in bold point out the specific contributions that were adapted for chapters of this dissertation.

Work	**Contribution(s)**	**Figure 1.4 Concept Vertex/Vertices**
[72]	Deep learning for approximate offset-free nonlinear MPC in embedded applications	digital, physical
[73]	Data-driven adaptation of control policies under model uncertainty	digital
[74]	Robust MPC with embedded adaptive scenario trees using Bayesian neural networks	digital
[75]	Extension of [74] with robust Bayesian neural networks	digital
[76]	Personalized medicine via adaptation of deep learning-based approximate MPC	digital, physical
[77]	Safe exploration for adaptation of (robust) MPC policies	digital
[78]	End-to-end hardware-software co-design for embedded control policies in plasma medicine	digital, physical
[79]	Identification and characterization of biological tissues using CAPs	biological
To be Submitted	Tunable biochemical outcomes in plasma activated water using multi-objective BO	digital, biological

Table 1.2: Summary of contributions of this thesis. The “Concept Vertex/Vertices” column relates back to Figure 1.4 and highlights the broad element(s) of design that each contribution falls under.

1.6 Organization

The remainder of this dissertation is organized as follows. **Chapter 2** describes the CAP jets (CAPJs) used in this dissertation. **Chapter 3** describes general problem formulations that are encountered throughout this dissertation. **Chapter 4** describes a multi-objective approach to the hardware-software design problem for CAPJs, particularly in exploring the connection between the digital and physical components of the control design process. **Chapter 5** discusses applications and extensions of Bayesian optimization to personalize plasma treatments and make control policy exploration safe without significant loss in performance gains. **Chapter 6** shifts the focus to the biological side and demonstrates how CAPs can be used to characterize biological materials. **Chapter 7** concludes this dissertation.

Chapter 2

Cold Atmospheric Plasma Jets

This chapter describes the cold atmospheric plasma jets used as experimental testbeds for prototypical applications in plasma medicine and/or plasma (bio)-processing. These testbeds were used to generate data to learn data-driven models used in in silico *closed-loop simulations and to demonstrate our Bayesian optimization framework in real-time experiments.*

2.1 Introduction

Cold atmospheric plasma jets (CAPJs) are a class of CAP-generating devices that are classified based on their electrode geometry and arrangement, the excitation frequency of the applied voltage, and the flow field configuration and composition of the working gas [80, 81]. This dissertation focuses on dielectric barrier discharge (DBD) jets and DBD-like jets, of which have several configurations that are illustrated in Figures 2 and 3 of [80]. DBD jets operate with kHz-frequency alternating current (AC) or pulsed direct current (DC) applied voltages applied to a noble working gas such as helium. DBD jets offer several advantages for plasma medicine [80]:

- The gas temperature remains close to room temperature ($\sim 30°$C) due to the low power density delivered to the plasma.
- The use of the dielectric ensures that there is no risk of arcing regardless of the tip-to-surface distance.
- The plasma plume and its effects can reach several centimeters beyond the tip of the plasma tube [82], which makes the operation of the plasma jet flexible and physically easy to manipulate.

DBD-like jets have similar advantages, but can operate in a non-DBD manner when the treated surface is conductive. Doing so increases the chance that an arc may occur, but can

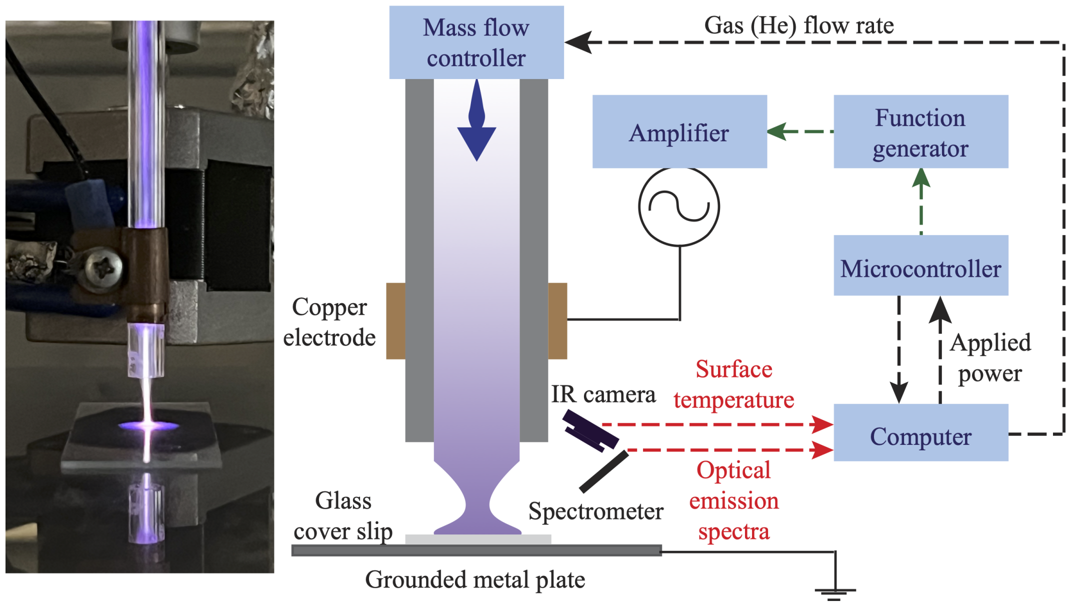


Figure 2.1: Close-up image (left) and schematic (right) of the kHz-excited CAPJ in helium (He). The manipulated inputs are denoted along the black dotted arrows, and the controlled outputs are denoted in red.

increase the overall power delivery to the plasma, which can create more rich chemistry. In plasma medicine, DBD-like jets would have to consider this particular trade-off.

This dissertation primarily uses a DBD jet in the configuration illustrated in Figure 2(b) of [80] with the exception of the final line of work described in Chapter 6, which uses a DBD-like configuration illustrated in Figure 3(a) of [80]. The following subsections give the details of the CAPJ testbeds.

2.2 Ring-Electrode kHz-excited Dielectric-Barrier Glow Discharge

The CAPJ testbed used in this dissertation for Chapters 4 and 5 was a part of prior lines of work in predictive control for CAPs [5, 6]. It is a DBD jet configuration that consists of a copper ring electrode wrapped around a quartz tube, which serves as a dielectric barrier and the gas flow channel. Ultra-pure helium is used as the working gas. An image and schematic of the CAPJ is shown in Fig. 2.1. Plasma ignition is achieved by applying a kHz-frequency, AC voltage to the copper electrode. The generated plasma is directed out of the tube onto a grounded, glass-covered metal plate at a distance of 4 mm below the tip of the tube. The applied voltage signal is created by generating a sinusoidal waveform at a specified frequency

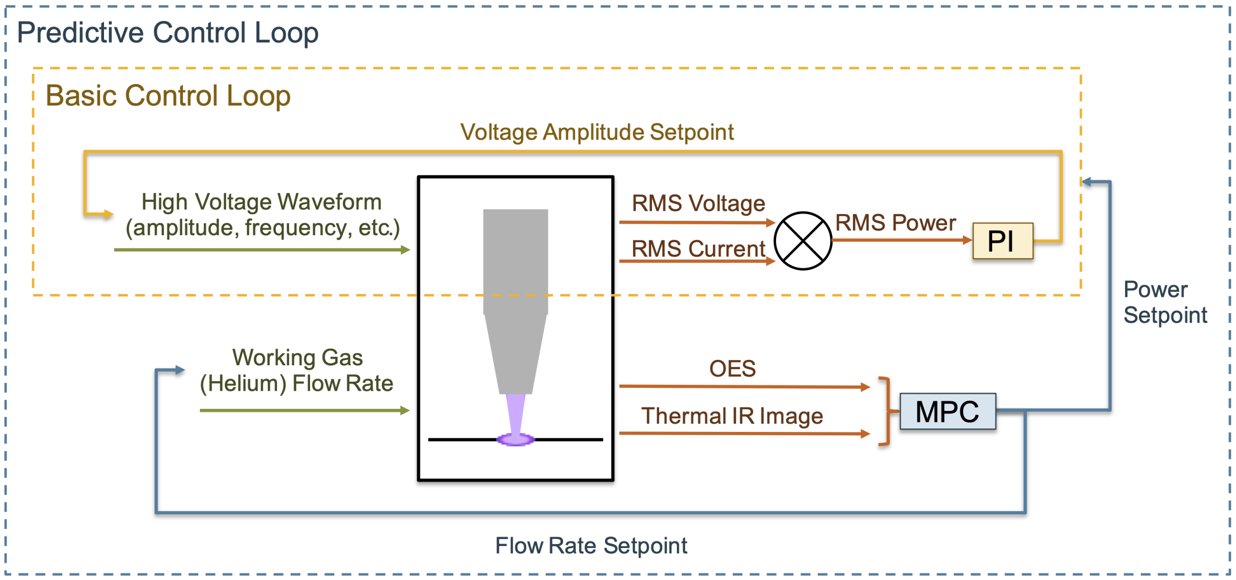


Figure 2.2: Illustration of the hierarchical control system for a cold atmospheric plasma jet (CAPJ) used in this dissertation. The green arrows represent the manipulated inputs of the CAPJ; the orange arrows represent the measured outputs of the CAPJ; the blue dashed box identifies the basic power control loop; and the yellow dashed box identifies the advanced predictive multi-output control loop.

using a function generator (integrated circuit, XR-2602CP). This signal is amplified using an amplifier (TREK 10/40A-HS) before being sent to the copper electrode. The flow rate of the working gas is manipulated by a UNIT Instruments UFC-1660 mass flow controller.

2.2.1 Control-oriented Testbed Setup

Automated data acquisition and actuation are necessary for the study of control systems. The CAPJ testbed by Gidon [5] used in this dissertation was the first of its kind to develop an automated sensing and actuation system for CAPJs. At the base level, the CAPJ has two manipulated inputs: the applied voltage waveform and the working gas flow rate. The applied voltage waveform can be further decomposed into characteristics of the wave (e.g., amplitude, frequency, type). Note that the composition of the working gas can also be manipulated, but is not the focus of this work. Then, a variety of thermal, electrical, and chemical sensing equipment can be used to take measurements of the CAP and of its interactions with surfaces.

In Chapters 4 and 5, we consider the CAPJ as a part of a hierarchical control system as illustrated in Figure 2.2. In this control configuration, there exist two feedback control loops: a basic proportional-integral (PI) control loop (dashed blue box) and an advanced model predictive control (MPC) loop (dashed yellow box). The basic control of this CAPJ setup involves the control of the applied power P_{app} to the plasma by manipulating the peak-to-peak amplitude of the voltage (V_{p2p}) waveform. Measurements of the voltage and current were taken using AD536A AC-to-RMS converters [5]. Basic control in the form of PI control is a powerful tool to create reproducible and reliable CAP effects, but cannot effectively address the multi-variable nature of the CAPJ operation [1]. A hierarchical control system involving a supervisory predictive control strategy was added to address the multi-variable operation of the CAPJ. This strategy was effective in controlling multiple outputs of the CAPJ relating to the chemical (via optical emission spectra (OES), $I(\lambda_{\text{oes}})$) and thermal effects (surface temperature, T_s) of CAP-surface interactions [61] by manipulating the applied power setpoint P and working gas flow rate q. Optical emission spectra were recorded using an Ocean Optics USB 2000+ spectrometer, and thermal data were recorded using FLIR Lepton 3 thermal infrared (IR) camera. For the majority of this dissertation, the basic control loop is fixed, and we consider the CAPJ system as a two-input, two-output system, with the applied power setpoint (P) and helium (working gas) flow rate (q) as the manipulated inputs and the surface temperature (T_s) and OES ($I(\lambda_{\text{oes}})$) as the controlled outputs.

2.2.2 Control-oriented Data-driven Modeling

A critical challenge in the development of model-based optimal control policies lies in the modeling of the CAP and its interactions with the target surface. CAPs are notoriously difficult to model since they exhibit nonlinear dynamics that are distributed over multiple length and time scales. Modeling difficulty is further exacerbated by the intrinsic variability in the plasma and sensitivity of the CAPJ to exogenous disturbances. Moreover, use of theoretical models [3, 83, 84] is ill-suited for real-time control of plasma effects that occur on the millisecond to second timescale. Instead, a common solution is to resort to data-driven modeling of the CAPJ [65, 85].

Commonly throughout this dissertation, we identify a linear, time-invariant (LTI) model using the `n4sid` function in MATLAB using input-output data of the CAPJ. Input-output data were gathered by performing multiple step tests in the inputs $u = [P, q]^\top$ and recording the outputs $y = [T_s, aI]^\top$, where a is a scaling factor to scale the total optical intensity to the same order of magnitude as surface temperature. Furthermore, the data were centered around nominal operating conditions $[P^s, q^s]^\top$ and $[T^s, I^s]^\top$, where the superscript s denotes the nominal condition. The model follows the discrete-time state-space form

$$x_{k+1} = Ax_k + Bu_k, \tag{2.1a}$$

$$y_k = Cx_k + Du_k, \tag{2.1b}$$

where $k \geq 0$ is the discrete time step, $x \in \mathbb{R}^{n_x}$ is the vector of states, $u \in \mathbb{R}^{n_u}$ is the vector of manipulated inputs, $y \in \mathbb{R}^{n_y}$ is the vector of measured outputs, and A, B, C, D are the state-space matrices identified using subspace identification [86]. The state-space model is defined in terms of deviation variables around the nominal operating condition, i.e., $y = [(T - T^s), (aI - I^s)]^\top$ and $u = [(P - P^s), (q - q^s)]^\top$. In this case, we assume an observable canonical form of (2.1), where $C = \boldsymbol{I}$ and $D = \mathbf{0}$ (e.g., see Appendix 4.8.1 for a description of exemplary model matrices). In the closed-loop simulation studies, the true system model of the CAPJ is treated as having a white noise term added to (2.1)

$$f(x_k, u_k, w_k) = x_{k+1} = Ax_k + Bu_k + w_k, \tag{2.2}$$

where w_k is generated from a uniform distribution with all elements bounded in $[-1, 1]$.

Plasma treatment not only depends on the current state of the plasma itself, but also on quantification of the delivered plasma effects to a surface. While quantification of plasma effects is generally cumbersome and application dependent [1], this dissertation takes inspiration from hypothermia treatments to quantify the delivery of a desired thermal effect (aka a thermal dose) [87]. A thermal dose metric is quantified in terms of cumulative equivalent minutes (CEM), which describes the accumulation of thermal effects on a target with respect to a reference temperature. The CEM is described by

$$\text{CEM}_{k+1} = \text{CEM}_k + K^{(T_{\text{ref}} - T_{s,k})}\delta t, \tag{2.3}$$

where $K = 0.5$ is an exponential base dependent on physical properties of the substrate, $T_{\text{ref}} = 43°\text{C}$ is the reference temperature, and δt is the sampling time. This definition of the thermal dose is cumulative, in that plasma effects delivered cannot be removed, and nonlinear due to the exponential dependence on temperature. Commonly throughout this dissertation, the control objective is to deliver a desired "dose" of thermal effects given by a target CEM value.

2.3 Coaxial Dielectric-Barrier Glow Discharge

In a collaborative effort, Chapter 6 uses a DBD-like CAPJ produced by the GREMI research group at the Université d'Orléans. Figure 2.3 shows a schematic and images of the CAP device and data acquisition setup used in Chapter 6. The CAP device configuration, known as the "Plasma Gun," is a coaxial dielectric barrier discharge jet using helium as the working gas. High voltage μs pulses (+7 kV peak amplitude with 600 Hz pulse repetition frequency using a custom power supply) are applied to an enclosed brass electrode, and an outer electrode surrounding the quartz capillary tube serves as a ground electrode. Once the high voltage pulse is applied, helium flowing (maintained at 0.5 SLM via a Bronkhorst EL-FLOW Prestige FG-201 CV) through the quartz capillary tube is ionized. The ionization front propagates outside the tube partially ionizing the surrounding ambient air. We maintained a distance of 5 mm between the tip of tube and the biological interface.

While control was not a focus of this subset of this chapter, an automatic data collection protocol was created for this setup in order to create data-driven characterization. As such, chemical and electrical data were collected via an Ocean Optics Maya 2000 Pro spectrometer and Picoscope 2406B oscilloscope, respectively, are connected to a computer to automatically obtain and record data at 0.5-second sampling intervals. A fiber optic cable, 45° from the tube axes and pointed at the plasma-tissue incidence point, is connected to the spectrometer and used to collect optical emission spectra. A compensation circuit [4] is used to mimic the electrical interactions with a non-human material to that of a human interface. Electrical characteristics of this system were taken at the locations marked by pentagons (A, B, and C) with voltage probes connected to the oscilloscope. Specifically, the applied high voltage was taken at location A using a Tektronix P6015A high voltage probe, and the voltage probes used at the ground electrode (location B) and after the compensation circuit (location C) are TA375 100 MHz probes and were connected across 200 pF capacitors in series to the ground.

The use of this CAPJ setup involved interactions with biological materials (specifically, raw chicken legs); the roast chicken in the schematic is for illustrative purposes only. Chicken legs were acquired the same day or the day before the testing and conserved at 4°C up to the moment of use. The top right image shows the data collection setup. The bottom right images illustrate the plasma interacting with the chicken leg (bone, left; muscle, right) during data collection.

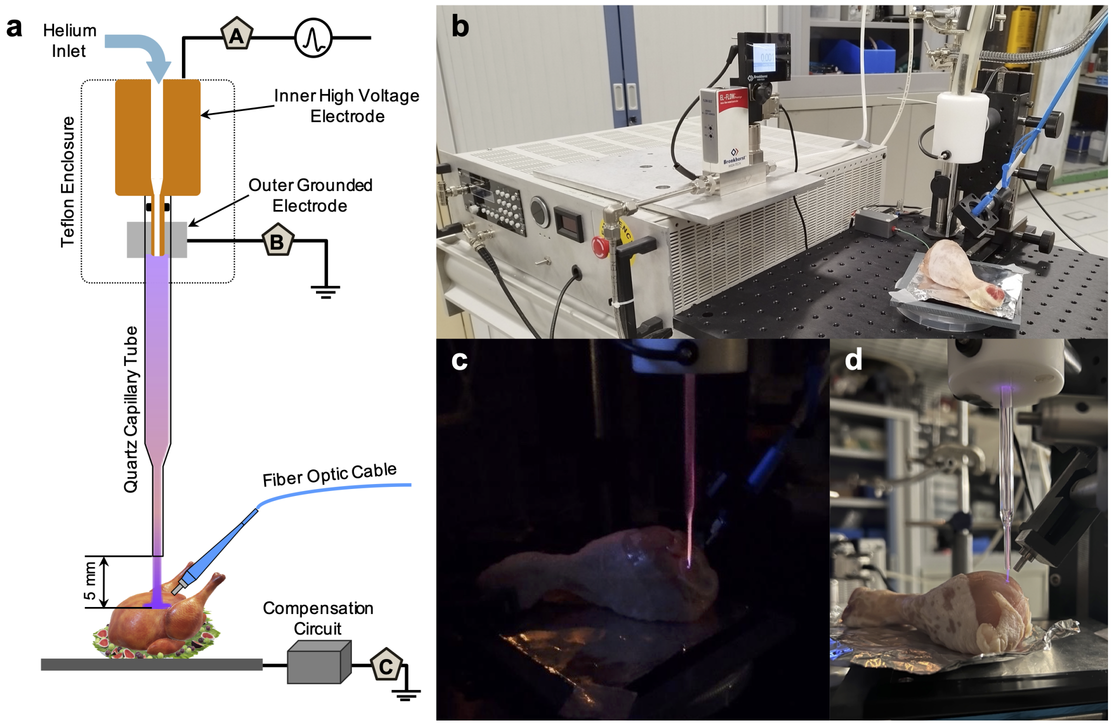


Figure 2.3: Cross-section schematic (**a**) and images (**b-d**) of the plasma gun setup. (**a**) A cross-section of the cold atmospheric plasma device that consists of a coaxial dielectric barrier discharge configuration using helium as the working gas. High voltage μs pulses (+7 kV peak amplitude with 600 Hz pulse repetition frequency via a custom power supply) are applied to an enclosed brass electrode, and an outer electrode surrounding the quartz capillary tube serves as a ground electrode. The plasma is generated by ionizing the helium flow in the quartz capillary, and the excited helium transfers energy to surrounding air constituents that make up the plasma plume impinging the samples. We maintain a distance of 5 mm between the tube end and the biological interface. A fiber optic cable pointed at the plasma-tissue incidence point and connected to a spectrometer (not shown) is used to collect optical emission spectra. A compensation circuit [4] is used to mimic the electrical interactions with a non-human material to that of a human interface. Electrical characteristics of this system were taken at the locations marked by pentagons (A, B, and C) with voltage probes connected to an oscilloscope (not shown). In Chapter 6, we use a raw chicken leg model to test various biological tissues, and the roast chicken in the schematic is for illustrative purposes only. (**b**) An image of the data collection setup; the plasma is powered by a custom power supply (large beige box on the left side of the image) and the flow rate of the helium is controlled by a mass flow controller (sitting on top of the power supply). (**c,d**) Images of the plasma interacting with the chicken leg (bone, (**c**); muscle, (**d**)).

Chapter 3

Optimal Control Formulations for Run-to-Run Plasma Treatment Regimens

This chapter describes the basic mathematical formulation of the control problems considered throughout this dissertation. This chapter aims to define the control problem(s) in a broad context as they relate to plasma treatments in medicine. Additional details/specific problems addressed are described in subsequent chapters.

3.1 Formulation of a Plasma Treatment

Consider a plasma treatment as the evolution some physical dynamics defined by:

$$x_{t+1} = f(x_t, u_t, w_t), \tag{3.1}$$

$$y_t = h(x_t, u_t, v_t), \tag{3.2}$$

where x_t, u_t, and y_t are the states, inputs, and outputs of the system at time t, w_t is some process noise at t, v_t is some measurement noise at t, $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_w} \to \mathbb{R}^{n_x}$ is the state evolution, and $h : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ is the output equation. For a predetermined treatment time T, the trajectories can be collected as a sequence of the state, input, process noise, output, and measurement noise realizations denoted as

$$X = (x_0, x_1, \dots, x_T), \tag{3.3}$$

$$U = (u_0, u_1, \dots, u_{T-1}), \tag{3.4}$$

$$W = (w_0, w_1, \dots, w_{T-1}), \tag{3.5}$$

$$Y = (y_0, y_1, \dots, y_T), \tag{3.6}$$

$$V = (v_0, v_1, \dots, v_T), \tag{3.7}$$

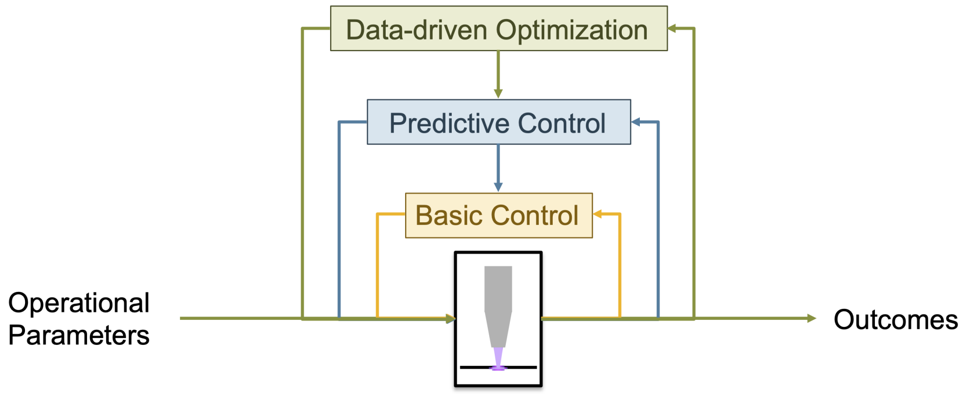


Figure 3.1: Hierarchical control with data-driven optimization as an additional layer of control. Prior works [5, 6] focused on the basic control and predictive control levels; this work focuses on the third level of control based on data-driven optimization. A timescale separation determines the need for this form of hierarchical control. The basic control layer operates at μ-second to millisecond timescales; the predictive control operates at millisecond to second timescales; and the data-driven optimization operates run-to-run at the minutes (or larger) timescale.

and the effect of a plasma treatment can be considered a function ψ of the trajectories

$$\psi(Y), \tag{3.8}$$

where Y is uniquely determined by X, U, W, V. For a *controlled* plasma treatment, u_t is determined by some parameterized control policy $\pi(x_t; \theta)$, where θ are the parameters that define the control policy and $\pi : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_u}$ represents a state-dependent feedback control policy. Combining a control policy π with the dynamics (3.1) creates trajectories that can be considered as random variables whose distributions depend on the choice of control policy parameters θ. Therefore, the effects of the plasma treatment should be evaluated as the expected value

$$J = \mathbb{E}\left\{\psi(Y(\theta))\right\}. \tag{3.9}$$

3.2 Predictive Control Policies

This section aims to formulate a generic structure for a broad class of predictive control strategies that exist for controlled plasma treatments, i.e., formulate $\pi(x_t; \theta)$. This level of control exists to reliably deliver multi-variable effects over the period of one plasma treatment as justified in prior work [5].

In this dissertation, we are interested in a particular subset of optimization-based control policies called model predictive control (MPC) [27, 28]. MPC and its variants are commonly used in multi-variable, complex systems to provide an interpretable or understandable set of equations for a control system, since its formulation relies on the specification of a control objective/cost and a set of physical constraints [69]. A generic MPC formulation may follow

$$\pi(x_t;\theta) = \arg\min_{u_{0|t}} \sum_{k=0}^{N_p-1} V(x_{k|t}, u_{k|t}, \hat{w}_{k|t}) + V_f(x_{N_p|t}), \tag{3.10a}$$

$$\text{subject to} \quad x_{k+1|t} = f(x_{k|t}, u_{k|t}, \hat{w}_{k|t}), \tag{3.10b}$$

$$x_{0|t} = x_t, \tag{3.10c}$$

$$x_{k|t} \in \mathcal{X}, \quad k = 0, \ldots, N_p, \tag{3.10d}$$

$$u_{k|t} \in \mathcal{U}, \quad k = 0, \ldots, N_p - 1, \tag{3.10e}$$

where $N_p \geq 1$ is the prediction horizon; $x_{k|t}$, $u_{k|t}$, and $\hat{w}_{k|t}$ are the predicted states, inputs, and disturbances, respectively, k steps ahead of the current time t; $\mathcal{X}$ and $\mathcal{U}$ are the sets of state and input constraints, respectively; and $V : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_w} \to \mathbb{R}$ and $V_{N_p} : \mathbb{R}^{n_x} \to \mathbb{R}$ are the stage and terminal costs, respectively. Note that the optimization problem includes the decision variables $x_{0|t}, \ldots, x_{N_p|t}$ and $u_{0|t}, \ldots, u_{N_p-1|t}$, but the control policy is defined with respect to the arg min over $u_{0|t}$ since MPC only applies the first optimal input.

This dissertation used existing formulations of MPC for the predictive control layer of hierarchical control, since the focus of this work lies in the data-driven optimization framework. Importantly, the data-driven optimization framework will rely on the update of parameters θ to tailor the outcomes of plasma treatment J. The purpose of defining the MPC structure here allows us to provide examples of control policy parameters θ based on MPC policies. Parameters of MPC policies that may be modified in a data-driven manner include $\{N_p, f, V, V_{N_p}, \mathcal{X}, \mathcal{U}, \hat{w}_{0|t}, \ldots, \hat{w}_{N_p-1|t}\}$. We note that a variant of robust MPC was explored in the context of probabilistic safety guarantees during a plasma treatment in [74] and [75].

3.3 Data-driven Optimization to Design Plasma Treatment Regimens

This section aims to describe the algorithmic details of Bayesian optimization (BO). Recall the optimization problem introduced in Chapter 1.4

$$\min_{\theta} \ \left\{ \{J_i(\theta)\}_{i=1}^{m} \quad \text{subject to } \{c_j(\theta)\}_{j=1}^{p} \geq 0 \right\}, \tag{1.1 revisited}$$

where J_i are defined as the effects of a plasma treatment (3.9) and the constraints c_j can be defined similarly. As mentioned in Chapter 1.4, (1.1) cannot be solved with standard optimization strategies (i.e., gradient descent) due to the black-box nature of J_i and c_j. Instead, we use BO, which is a derivative-free optimization method with low resource utilization. BO recasts an optimization problem into a sequential learning problem and consists of two main components: (i) a predictive probabilistic surrogate model representing the objective(s) and constraint(s) and (ii) an “acquisition” function that quantifies the benefit of querying the posterior objective(s) and constraint(s).

Create a Predictive Probabilistic Surrogate Model

To start, BO requires a predictive model defined by a Bayesian posterior distribution over J. Some examples of probabilistic surrogate models that satisfy this requirement include random forests [88], Bayesian neural networks [89], and Gaussian processes (GP) [90]. GPs are the most commonly used class of models in BO due to their computationally tractability and non-parametric nature [90]. A GP is fully specified by its prior mean $\mu_0 : \Theta \to \mathbb{R}$ and covariance $k_0 : \Theta \times \Theta \to \mathbb{R}$ functions. New data are obtained by querying the true system $\mathcal{D}_n = \{(\theta_j, \{y_{i,j}\}_{i=1}^{M})\}_{j=1}^{n}$, where $y_{i,j} = J_i(\theta_j) + \varepsilon_{i,j}$ are noisy evaluations with independent and normally distributed, zero-mean, σ^2-variance noise $(\varepsilon_{i,1}, \dots, \varepsilon_{i,n})$. Then, the posterior distribution for each J_i can be computed with the following analytic mean and covariance expressions

$$\begin{aligned} \mu_n(\theta) &= \mu_0(\theta) + \boldsymbol{k}_0^\top(\theta)(\boldsymbol{K}_n + \sigma^2 I_n)^{-1}\tilde{\boldsymbol{y}}_n, \\ k_n(\theta, \theta') &= k_0(\theta, \theta') - \boldsymbol{k}_0^\top(\theta)(\boldsymbol{K}_n + \sigma^2 I_n)^{-1}\boldsymbol{k}_0(\theta), \end{aligned} \tag{3.11}$$

where $\tilde{\boldsymbol{y}}_n = (y_1 - \mu_0(\theta_1), \dots, y_n - \mu_0(\theta_n)) \in \mathbb{R}^n$, $\boldsymbol{k}_0(\theta) = (k_0(\theta, \theta_1), \dots, k_0(\theta, \theta_n)) \in \mathbb{R}^n$, $\boldsymbol{K}_n \in \mathbb{S}_{++}^n$ is the kernel matrix whose elements are given by $[\boldsymbol{K}_n]_{\nu,\omega}$ for all $\nu, \omega \in \{1, \dots, s\}$, and I_n is the $n \times n$ identity matrix. The key advantage behind this probabilistic predictive model lies in the fact that the posterior mean function μ_n can be interpreted as a surrogate model for the closed-loop performance measures J_i and that the posterior covariance function k_n provides rigorous uncertainty estimates. These estimates are directly used in the next component of BO.

Optimize an Acquisition Function

The exploration-exploitation dilemma is a long-standing concept in decision-making [91]. The core principle is that in order to maximize long-term benefit (or minimize long-term cost), there exists some balance between choosing the next action based on past experience/information (exploitation) or based on the prospect of gaining more information in the future (exploration). BO accomplishes this by recasting the main optimization to solve the following optimization problem

$$\theta_{n+1} \in \operatorname*{arg\,max}_{\theta \in \Theta} \boldsymbol{\alpha}_n(\theta), \tag{3.12}$$

where $\boldsymbol{\alpha}_n : \Theta \to \mathbb{R}$ is an acquisition function that quantifies the exploration-exploitation tradeoff. Conceptually, the acquisition function is derived from an expected increase in some utility (or reward) function $r(\mathcal{D}_n)$ dependent on the observed data $\mathcal{D}_n = \{\theta_j, \{y_{i,j}\}_{i=1}^{M}\}_{j=1}^{n}$, where $y_{i,j} = J_i(\theta_j) + \varepsilon_{i,j}$ are noisy evaluations with independent and normally distributed,

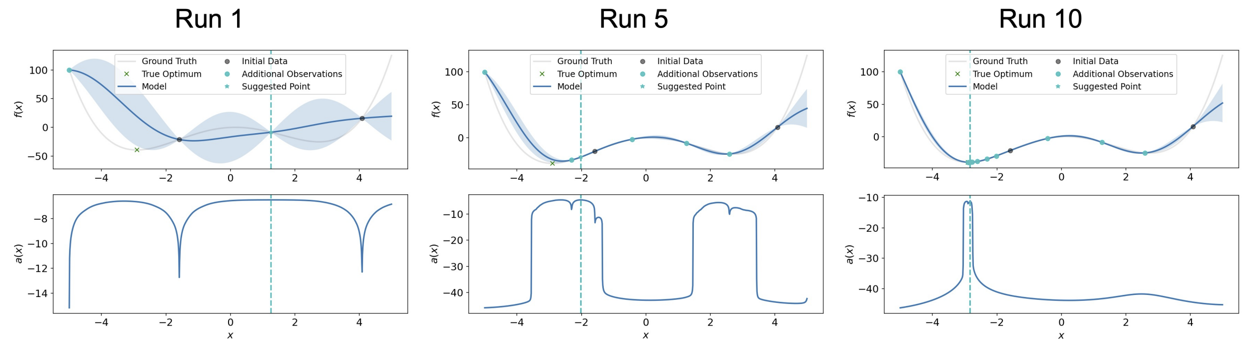


Figure 3.2: Evolution of Bayesian optimization for a one-dimensional single objective problem. The top subplots illustrate the Gaussian process surrogate model and how it evolves in time from left to right. As data is observed, the model more accurately represents the truth, and the uncertainty around known data is reduced. The bottom subplots show the acquisition function and how it evolves in time from left to right. The value of searching in particular points is highlighted by the acquistion function.

zero-mean, σ^2-variance noise $(\varepsilon_{i,1}, \ldots, \varepsilon_{i,n})$, i.e.,

$$\boldsymbol{\alpha}_n(\theta) = \mathbb{E}_n \left\{ r(\mathcal{D}_n \cup (\theta, \{y_i\}_{i=1}^{M})) - r(\mathcal{D}_n) \right\}. \tag{3.13}$$

It is important to note that the acquisition function $\boldsymbol{\alpha}_n$, unlike the J_i's, is often available in closed-form, making (3.12) easier to solve. The main principle behind α_n is that it considers both the posterior mean μ_n (belief about the true system) and the posterior covariance k_n (or, more commonly, the variance $\sigma_n = \sqrt{k_n}$, the belief about the uncertainty). As such, it trades off between optimizing the belief about the true optimum (exploitation) and the belief about potential optimums (exploration). This trade-off can be interpreted as a formal way to quantify the information gained from testing a set of control policy parameters defined by θ_n. In a single objective case, the value of information is placed on locating one true optimum. An illustration of the evolution of BO for a single objective is shown in Figure 3.2. In a multi-objective case, the value of information is placed on finding the Pareto optimal points (aka the points at which optimizing one measure degrades the other). The multi-objective case ultimately results in a more general representation of possible controller configurations, allowing the user to select the best fit for the application based on the Pareto optimal points. Details regarding multi-objective optimization will be discussed in subsequent chapters.

Chapter 4

Multi-Objective Learning Framework for Optimal Hardware-Software Co-Design of Control-on-a-Chip Systems[1]

The digital age has made embedded control a key component to user-oriented, portable, and internet-of-things devices. In addition, with emergent complex systems arises the need for advanced optimization-based control strategies like model predictive control. However, the unified implementation of these advanced strategies on hardware remains a challenge. Designing complex control policies for embedded systems is inherently an interwoven process between the algorithmic design and hardware implementation, which will require a hardware-software co-design perspective. We propose an end-to-end framework for the automated design and tuning of arbitrary control policies on arbitrary hardware. The proposed framework relies on deep learning as a universal control policy representation and multi-objective Bayesian optimization (BO) to facilitate iterative systematic controller design. The large representation power of deep learning and its ability to decouple hardware and software design are a central component to determining feasible control-on-a-chip policies. Then, BO provides a flexible sequential decision-making framework where practical considerations such as multi-objective optimization concepts and categorical decisions can be incorporated to efficiently design embedded control policies that are directly implemented on hardware. We demonstrate the proposed framework via closed-loop simulations and real-time experiments on an atmospheric pressure plasma jet for plasma processing of biomaterials.

[1]This chapter was adapted with permission from the coauthors from [78].

4.1 Introduction

Embedded systems lie at the core of many online control systems technologies, including autonomous systems [92], IoT systems [93, 94], and biomedical devices [95, 96], amongst others. Microcontrollers/microprocessors (MCUs) have played a major role in enabling embedded control for portable devices [97]. Due to their widespread availability and adoption, MCUs have been designed such that they can be easily programmed using high-level programming languages, such as C and Python, and be deployed with relative ease [93]. Meanwhile, emerging technologies have increasingly complex dynamics and typically rely on advanced model-based control strategies, such as model predictive control (MPC), that can handle constraints. As a result, significant efforts have targeted the development of automated software-based code generation tools for fast numerical optimization on MCUs. These tools (e.g., ACADO [98], GRAMPC [99], FORCES [100]) utilize tailored implementations of structured formulations of the underlying optimization problem to efficiently compute the optimization solution for real-time control (e.g., using sequential quadratic programming [101], nonlinear interior point methods [102], or accelerated gradient methods [103]). The structured formulation that these code generation tools require can make the implementation of robust and learning-based optimal control strategies more challenging, while the focus on solely the software side implies that the form of hardware implementation is also limited. Furthermore, the design of embedded controllers relies on more than just the fast solution of the optimal control problem. It also involves maintaining numerical robustness at low computational accuracy, tolerance against infeasibility, low code complexity, and low memory/resource utilization, among other considerations. Many of these challenges require explicit knowledge of the hardware specifications, as well as knowledge of how computations are performed and accelerated on the hardware. As advanced hardware technologies (e.g., graphics processing units (GPUs), field programmable gate arrays (FPGAs), tensor processing units (TPUs)) become commonplace, the principle of *hardware-software co-design* will become an integral consideration for the physical implementation of embedded controllers [104]. Hardware-software co-design involves the concurrent design of the control algorithm (software) and its embedded implementation (hardware) [105].

Optimized control policy tuning is often a tedious and cumbersome process, which is further exacerbated by the extensive workflow to go from the programmatic control policy design to embedded implementation. Control policy auto-tuning (aka calibration) is well-established for simple controllers [106,107], but auto-tuning for generic control structures has recently regained traction. One popular approach to auto-tuning uses principles of reinforcement learning (RL) [108–110]. To this end, policy-gradient RL methods involve updating the control policy parameters via gradient descent [111]. While policy gradient is a scalable approach, it can require many evaluations on the true system and can be prone to getting stuck at local optimizers. Another popular approach to auto-tuning relies on data-driven optimization, particularly Bayesian optimization (BO) [69, 112, 113]. Auto-tuning can be interpreted as a black-box problem where the objective function is expensive to evaluate, potentially non-convex, and without closed-form derivatives. BO is a "global" optimization method that takes a probabilistically principled approach to reduce the number of interactions with the real system [70]. Variants of BO for auto-tuning are also emerging [114–116]; however, most of these studies, except [115], do not consider the hardware considerations for embedded control. Even still, the focus of [115] remains on the hardware-constrained optimization (i.e., real-time computations) of the software, rather than hardware-software co-design.

The hardware-software co-design paradigm presents a new take on the embedded control design problem. Formally, we denote this new perspective of hardware-software co-design as control-on-a-chip (CoC) design, since we aim to provide a unified workflow to place arbitrary control policies on arbitrary hardware. In this work, we pose the CoC co-design problem as an optimization problem that incorporates a hardware feasibility constraint and multiple levels of decisions to create an all-in-one framework. First, we establish a flexible workflow that takes advantage of recent advances in deep learning. Deep neural networks (DNNs) have played a pivotal role in imitation learning of MPC policies [76, 117, 118] and differentiable predictive control [119]. In this work, we use the imitation learning perspective as it poses two key advantages: (i) it provides a "physically interpretable" control policy; and (ii) by virtue of (i), it reduces the design parameter space. Thus, our proposed CoC workflow consists in (i) designing an expert control policy (based on optimization-based control strategies), (ii) using DNNs to represent the expert control policy, and (iii) implementing the DNN-based control policy on hardware. We then use multi-objective BO (MOBO) [120] to encapsulate the multi-step CoC design workflow to create an end-to-end optimization framework. Solving a multi-objective CoC design problem via MOBO entails finding an optimal set of control policies rather than one single optimizer, which allows a practitioner to choose the best design(s) according to the needs or preferences of the application [121, 122]. Furthermore, BO offers flexibility when incorporating design choices from each step of the CoC workflow [90, 123].

4.2 Hardware-Software Co-design Problem Formulation

In this work, we seek optimal CoC policies that minimize a set of closed-loop cost metrics subject to hardware constraints. In general, CoC policies can be represented as a space of all possible machine code instructions that can be executed on a given choice of hardware. Since this is a complex design space involving decisions made by human experts, the design problem is often decomposed into two key steps: (i) choice of a high-level control program θ that must reside in the space of possible programs Θ (generally very high-dimensional and complex); and (ii) choice of a code generation strategy that translates θ into executable machine code, which has its own set of design parameters denoted by $\gamma \in \Gamma$ (e.g., numerical representation and parallelization options). We can formulate the search for an optimal pair of program and code generation parameters $(\theta^\star, \gamma^\star)$ in terms of the following multi-objective optimization (MOO) problem

$$\begin{aligned}
\min_{\theta,\gamma} \quad & \{J_1, \dots, J_M\}, && \text{(4.1a)}\\
\text{s.t.} \quad & x_{k+1} = f(k, x_k, u_k, w_k), && \text{(4.1b)}\\
& u_k = \pi(x_k; \theta, \gamma), && \text{(4.1c)}\\
& w_k \sim P_{w_k}(x_k, u_k), && \text{(4.1d)}\\
& J_i = \mathbb{E}\left\{ \ell_{T,i}(x_T) + \textstyle\sum_{k=0}^{T-1} \ell_i(x_k, u_k, w_k) \right\}, && \text{(4.1e)}\\
& g(\theta, \gamma) = 1, && \text{(4.1f)}\\
& (\theta, \gamma) \in \Theta \times \Gamma, && \text{(4.1g)}\\
& \forall (k, i) \in \{0, \dots, T-1\} \times \{1, \dots, M\},
\end{aligned}$$

where (4.1a) represents the set of M closed-loop performance metrics; (4.1b) represents the system dynamics that describes the evolution of the system state x_k in response to control actions u_k and disturbances w_k at time k; (4.1c) defines the CoC policy $\pi(x_k; \theta, \gamma)$ that maps (measured or estimated) states to control actions for a specific choice of program and code generation parameters; (4.1d) is a stochastic disturbance that evolves according some probability distribution $P_{w_k}(x_k, u_k)$ that is conditionally independent of previous disturbance realizations given the current states and actions; (4.1e) represents the i-th performance metric J_i defined in terms of the closed-loop system evolution given local stage cost $\ell_{k,i}$ and terminal cost $\ell_{T,i}$ functions over a finite time horizon T; (4.1f) denotes a hardware resource utilization constraint represented by a binary function $g : \Theta \times \Gamma \to \{0, 1\}$ that indicates if the high-level control program can be compiled and executed on the available hardware (1) or not (0); and (4.1g) represents the user-defined search space of control programs and code generation strategies. The expectation $\mathbb{E}\{\cdot\}$ in (4.1e) is taken with respect to the stochastic disturbance sequence of the closed-loop system $\{w_0, \dots, w_{T-1}\}$.

The MOO problem (4.1) represents a very general framework for CoC design. In fact, we can interpret virtually all end-to-end control design procedures as an approximation to (4.1). However, approximations are necessary in practice due to the intractability of (4.1), which stems from two main challenges. First, the program parameter space Θ is abstract due to the choice of an appropriate control policy representation. To ensure it has sufficiently large representation power, the control policy will typically be embedded in some high-dimensional space $\Theta \subset \mathbb{R}^D$, where D can be very large. Furthermore, this space may involve discrete and continuous variables that are needed to represent logical relationships between different sets of variables. Second, we often do not have exact knowledge of the system dynamics $f(\cdot)$, disturbance distribution $P_{w_t}(x_t, u_t)$, and the hardware utilization constraint function $g(\cdot)$, which prevents the application of traditional MOO methods that require equation-oriented forms for all objective and constraint functions.

Standard control design approaches often proceed in the following steps that are illustrated in Fig. 4.1: (i) restrict the control policy representation by constraining Θ to describe a narrow set of policies in a low-dimensional space; (ii) independently search for control pro-

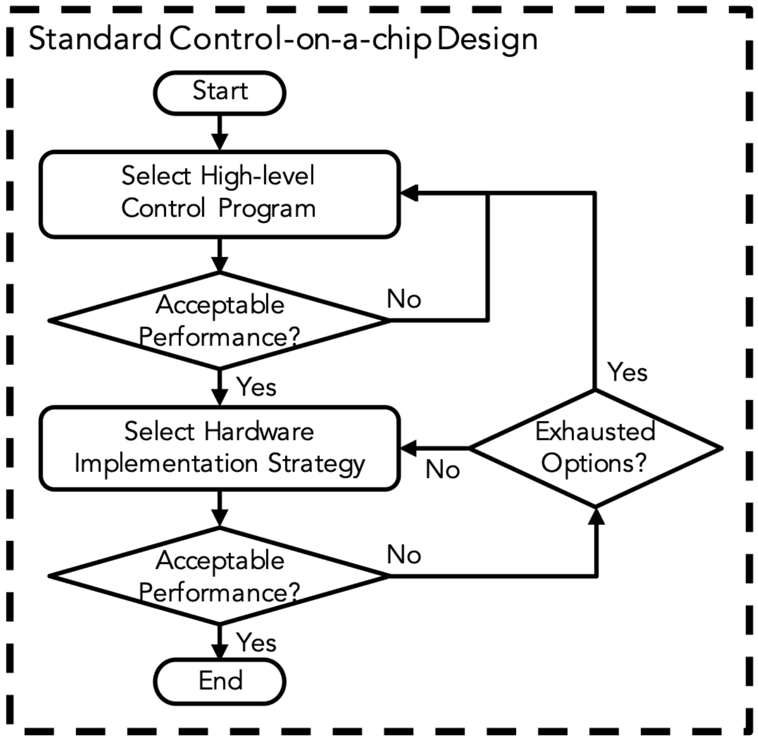

Standard Control-on-a-chip Design
Start
Select High-level Control Program
Acceptable Performance?
No
Yes
Select Hardware Implementation Strategy
No
Exhausted Options?
Yes
Acceptable Performance?
No
Yes
End

Figure 4.1: Flow diagram of the standard control-on-a-chip design process. First, a high-level representation of the control policy is selected and evaluated using approximate models or limited closed-loop data. Then, a code generation strategy is selected and evaluated based on its ability to be successfully implemented while matching the performance of the high-level program. In general, several iterations may be needed at each stage of the design process until an acceptable option is found. If any stage fails, then one must return to a previous stage to repeat the process.

gram parameters θ that approximately minimize the closed-loop performance metrics either using approximate models, or (limited) closed-loop data; and (iii) search for code generation parameters γ that enable the desired θ to be executed on the available hardware. If step (iii) fails, then one must go back and repeat the process again for a narrower set of more computationally tractable policies. For example, complex control policy formulations as in optimization-based control (e.g., as described in Section 4.3.3, (4.5)) may require specific code libraries and/or routines that are not easily implemented on low-resource hardware due to memory restrictions, or reduced accuracy due to quantized numeric representation. Furthermore, emergence of advanced hardware (e.g., GPUs, FPGAs) to speed up computations requires additional low-level code generation that requires specialized knowledge of the device architecture. Separate design of θ and γ misses out on important interactions between high-level program representation and code generation. In particular, the hardware utilization constraint function $g(\cdot)$ represents whether or not the policy can be embedded into the hardware or can satisfy timing constraints. Examples of parameters that influence this constraint include numeric representation, number of operations, parallelization options, etc. Without co-design, there is a much greater chance that CoC policies cannot be implemented as knowledge of how the algorithmic requirements are translated to the physical wires in hardware are not generally accessible.

The main goal of this work is the development of an iterative learning-based strategy capable of systematically tackling the *hardware-software co-design* problem (4.1). The core structure of our proposed strategy, which relies on deep learning to simplify the policy and hardware utilization constraint, is presented in the next section. In Section 4.4, we then describe an efficient multi-objective black-box optimization strategy that takes advantage of this structure by searching over a reduced set of parameters.

4.3 Bridging the Gap between Hardware and Software with Deep Learning

The lack of an easy-to-search program space Θ and known structure for the feasibility constraint g greatly complicates the traditional CoC design procedure (Fig. 4.1). In this section, we show how both of these problems can be addressed by working with deep neural network (DNN) policies such that $\theta = \{\theta_W, \theta_A\}$ can be separated into continuous weight and bias parameters θ_W and architecture parameters θ_A that can be discrete. This not only helps us simplify the learning process for g, but also allows us to take advantage of prior knowledge to "train" θ_W such that we only consider a small subset of parameters when optimizing closed-loop performance.

4.3.1 Deep Learning for Control Policy Representations

Deep learning is a generalized term for computational structures/graphs characterized by multiple "layers." Through multiple layers, deep learning transforms an input representation to abstract representations until the ultimate output is learned [124]. It is exactly the many layers that allow deep learning to extract (on its own) features from raw data that are relevant to learning control policies [2, 118].

The key advantage of DNN policies is the surprisingly robust ability to train such large structures using (stochastic) gradient descent style methods. For simplicity of presentation, consider a fully-connected feedforward DNN control policy $\pi_{\text{dnn}}(x;\theta)$ with L hidden layers and H nodes per layer, which can be mathematically defined as follows

$$\pi_{\text{dnn}}(x;\theta) = \alpha_{L+1} \circ \beta_L \circ \alpha_L \circ \cdots \beta_1 \circ \alpha_1(x), \tag{4.2}$$

where $\alpha_1(x) = W_1 x + b_1$ is an affine transformation of the input, $\alpha_l(z_{l-1}) = W_l z_{l-1} + b_l$ are affine transformations of the hidden layers for all $l \in \{2, \dots, L+1\}$, $\beta_l(z)$ are nonlinear activation functions (e.g., $\beta_l(z) = \max\{z, 0\}$ for ReLU activation functions) for all $l \in \{1, \dots, L\}$, and $\theta_W = \{W_1, b_1, \dots, W_{L+1}, b_{L+1}\}$ denotes the collection of weights and biases that parameterize the network for a fixed architecture θ_A (e.g., type of activation function, L, and H). Due to their continuous representation, θ_W can be trained by minimizing a loss function that captures how well the DNN performs on a given task. This process is known to work well in practice under the assumption that the gradient of the loss function with respect to θ_W can be efficiently computed via backpropagation [125]. This is not necessarily the case when one attempts to use π_{dnn} in (4.1) unless a differentiable structure for the dynamics and cost functions is known.

Remark 1 *Note that the DNN policy defined in* (4.2) *is just one choice of architectural representation of artificial neural networks. In fact, any deep learning architecture can be used to approximate* (4.2). *For example, if the state or any other exogenous signals involved image data, we could exploit a convolutional neural network structure that is designed to specifically exploit the regularity of image pixel patterns.*

4.3.2 Learning Feasible Space of CoC Policies

Given that our ideal policy is represented by a DNN, in addition to the program (i.e., software) (4.2), CoC policies also require specification of the embedded version of the program that can be run on the actual hardware. Although this difference is not practically important in the absence of resource limitations, it is very important in cases where there are constraints on the number of real-time computations and/or resource (memory or power) utilization [126]. A useful property of DNNs is that the resource utilization is the same for all θ_W given a fixed architecture θ_A. To see this, we can compute the number of operations N_{op} for a dense DNN of the form (4.2) as

$$N_{\text{op}} = (n_{\text{in}} + 1)H + H(H + 1)L + (H + 1)n_{\text{out}}, \tag{4.3}$$

where n_{in} and n_{out} are the number of inputs and outputs, respectively. Thus, by simply changing the architecture of the DNN (e.g., reduce number of nodes or layers), we can lower the evaluation cost on hardware.

Nonetheless, we cannot use N_{op} to directly characterize the feasible set of CoC policies, i.e., $\mathcal{F} = \{(\theta, \gamma) \in \Theta \times \Gamma : g(\theta, \gamma) = 1\}$, since this set will depend on how the operations written in a mid- or high-level programming language (e.g., Matlab, Python, C) get translated to low-level machine code (e.g., assembly language, binary), compiled, and then packaged. Automatic code generation tools aim to provide a streamlined means of performing such tasks by abstracting the laborious translation process [32, 127, 128]. As such, code generation serves as a bridge between human-interpretable code and machine-interpretable instructions. In this work, we treat code generation as a black-box function

$$\pi(\cdot; \theta, \gamma) = \mathcal{CG}(\pi_{\text{dnn}}(\cdot; \theta), \gamma), \tag{4.4}$$

that takes as input a DNN policy and some parameters related to the translation process γ and returns a machine-interpretable policy. Since changing θ_W will not fundamentally change the structure of the returned CoC policy π, function g will be independent of θ_W and, thus, we only need to learn an approximation of $g(\theta_A, \gamma)$.

One way to learn an approximation $\tilde{g} \approx g$ is to run the code generation process $\mathcal{CG}$ for a randomly generated DNN for several values of $(\theta_A, \gamma) \in \Theta_A \times \Gamma$, which is expected to be a much lower-dimensional space than Θ_W. The outcome can be recorded as either a successful compilation 1, or failed compilation 0. This labeled data can then be used to train a binary classifier that is capable of predicting if a new choice of architecture and code generation parameters is feasible or not. To this end, any classifier type can be used, for example, support vector machines and DNN with a sigmoid activation function at the output layer. We highlight that the major advantage of this approach is that $\tilde{g}$ can be trained *independently* of the quality of the CoC policy. Not only can this process be done fully offline, but also it does not require any system data to be generated – all that is required is access to the hardware and code generation process.

4.3.3 Accelerated Training of Hardware-Feasible DNN Policies by Imitating Physics-Informed Expert Policies

In Section 4.3.2, we presented an efficient way to verify if a CoC policy will be feasible. Yet, the original MOO problem (4.1) can still be computationally intractable since θ_W remains a high-dimensional space with possibly thousands or more independent parameters. The question we address here is how the search over this space can be efficiently performed without sacrificing the achieved closed-loop performance. To this end, we rely on a class of control policies that are implicitly defined in terms of a set of interpretable set of equations [69]. Specifically, we look to use policies defined by an optimization problem

$$\begin{aligned} \pi_{\text{opt}}(x; \lambda) = \arg\min_{u} \quad & V(x, u; \lambda), && && (4.5a) \\ \text{s.t.} \quad & h_i(x, u; \lambda) \le 0, && i = 1, \dots, k, && (4.5b) \\ & g_i(x, u, \lambda) = 0, && i = 1, \dots, r, && (4.5c) \end{aligned}$$

where $V : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\lambda} \to \mathbb{R}$ is the objective function, $h_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\lambda} \to \mathbb{R}$ are the inequality constraints for all $i = 1, \dots, k$, $g_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\lambda} \to \mathbb{R}$ are the equality constraints for all $i = 1, \dots, r$, and $\lambda \in \Lambda \subset \mathbb{R}^{n_\lambda}$ are tunable policy parameters. As discussed in [69], this representation captures a large set of policies, including approximate dynamic programming and model predictive control. The key idea behind (4.5) is that (4.5a) describes some type of value or reward function, (4.5b) represents critical state and/or input constraints, and (4.5c) represents a (possibly physics-based) model of the system. A significant advantage of the structure (4.5) is that it provides a natural way for users to incorporate prior knowledge about the system, when available, by properly selecting or constraining the functions $V, h_1, \dots, h_k, g_1, \dots, g_r$.

The central notion is that λ can be much lower dimensional than θ_W, such that we can derive an explicit value for the DNN parameters that depend on λ by minimizing the error between the DNN policy π_{dnn} and the "physics-informed" expert policy π_{opt}

$$\theta_W^\star(\lambda, \theta_A) = \arg\min_{\theta_W} \frac{1}{n_s} \sum_{i=1}^{n_s} \|\pi_{\text{opt}}(x^{(i)}; \lambda) - \pi_{\text{dnn}}(x^{(i)}; \theta_W, \theta_A)\|^2, \tag{4.6}$$

where $\{(x^{(i)}, \pi_{\text{opt}}(x^{(i)}; \lambda))\}_{i=1}^{n_s}$ represent a set of n_s state-action pairs acquired by solving the optimization problem (4.5) offline for specific state values and fixed λ values. This dataset can be generated in a variety of ways, including randomly sampling in the state space or using closed-loop "rollouts" from likely initial conditions [2]. It is important to note that special care must be taken in generating the dataset to train the DNN as the approximation of (4.5) may reduce the robustness properties ensured by implementing (4.5) directly depending on the quality of the overall training process. In accordance with the universal approximation theorem [129], there exists a DNN that matches (4.5) exactly. As such, given a sufficiently large architecture and enough training data, we can ensure $\pi_{\text{dnn}}(\cdot; \theta_W^\star(\lambda, \theta_A), \theta_A) \to \pi_{\text{opt}}(\cdot; \lambda)$ for some θ_A.

Notice that the solution to (4.6) will depend on both the expert policy parameters λ and the DNN architecture hyperparameters θ_A. Therefore, the proposed CoC policy has the following unique structure

$$\pi_{\text{CoC}}(\cdot; \lambda, \theta_A, \gamma) = \mathcal{CG}(\pi_{\text{dnn}}(\cdot; \theta_W^\star(\lambda, \theta_A), \theta_A), \gamma), \tag{4.7}$$

which depends on three sets of parameters, mainly λ, θ_A, and γ that all appear in different components of the CoC framework. An illustration of the proposed CoC design process is shown in Fig. 4.2, which is based on selecting $(\lambda, \theta_A, \gamma) \in \Lambda \times \Theta_A \times \Gamma$ to optimize closed-loop performance metrics of interest. However, we do not have a closed-form expression for how performance metrics depend on $(\lambda, \theta_A, \gamma)$. This is further compounded by the cost of collecting closed-loop performance data since it requires (i) training a DNN policy, (ii) executing a code generation process to run the policy on embedded hardware, and (iii) running hardware-in-the-loop closed-loop experiments to collect performance data. Next, we present an efficient procedure for searching over this joint parameter space.

4.4 Bayesian Optimization for Control-on-a-Chip Problem

The system dynamics and CoC policy together form a stochastic process due to the initial condition x_0 and disturbances $\{w_k\}_{k\geq 0}$ that are random variables

$$x_{k+1} = f(k, x_k, u_k, w_k), \quad k = 0, 1, \ldots, \tag{4.8a}$$

$$u_k = \pi_{\text{CoC}}(x_k; \xi), \tag{4.8b}$$

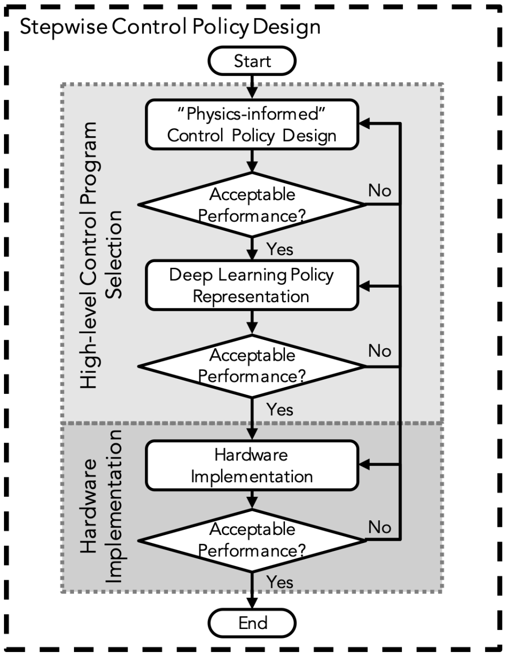

Stepwise Control Policy Design
Start
High-level Control Program Selection
"Physics-informed" Control Policy Design
Acceptable Performance?
No
Yes
Deep Learning Policy Representation
Acceptable Performance?
No
Yes
Hardware Implementation
Hardware Implementation
Acceptable Performance?
No
Yes
End

Figure 4.2: Flow diagram of the proposed control-on-a-chip (CoC) design process. As in Fig. 4.1, CoC design is subdivided into two categories related to software (high-level control program selection in light gray) and hardware implementation (in dark gray). Our proposed workflow for CoC design is subdivided as follows: Within the high-level control program selection, the first step is to select and evaluate a "physics-informed" control design. In the next step, a deep learning-based policy is created in pursuit of hardware-compatibility. The final step involves the hardware implementation and final evaluation. Note that the "Exhausted Options" decision marker is not present in this figure for simplicity, but still exists as part of the design process. We define this design process as a framework for CoC design that can be used to search over the joint software and hardware parameter space.

where $\xi = (\lambda, \theta_A, \gamma)$ is the concatenation of all software and hardware parameters that define the policy. A specific choice of ξ can be judged according to the set of M expected closed-loop performance metrics $J_i(\xi)$, $i = 1, \dots, M$ defined in (4.1) with the state and input sequences generated by (4.8).

Since J_i are defined as expectations over closed-loop trajectories, they only depend on ξ that is of much lower-dimensional than the original θ space, as discussed above. Thus, we now pose (4.1) as a more manageable MOO problem

$$\min_{\xi \in \Xi} \; \{J_1(\xi), \dots, J_M(\xi)\}, \tag{4.9}$$

where $\Xi = \{(\lambda, \theta_A, \gamma) \in \Lambda \times \Theta_A \times \Gamma : \tilde{g}(\theta_A, \gamma) = 1\}$ is the space of CoC parameters that can be compiled on the available hardware. Since the functions $\{J_i\}_{i=1}^{M}$ are black box in nature, we must resort to derivative-free optimization (DFO) methods to approximately solve (4.9) in practice. In particular, the DFO method must be able to handle noisy, expensive evaluations of $\{J_i\}_{i=1}^M$. The evaluations are expensive due to the need to collect hardware-in-the-loop data, as discussed previously. These evaluations will also be subject to noise due to the expectation operator that defines J_i. In practice, we can approximate this expectation using a random sampling technique (e.g., Monte Carlo sampling) such that $y_i = J_i(\xi) + \varepsilon_i$, where ε_i is the effective measurement noise in the i-the performance function. Assuming K independent random samples are used to approximate the performance functions, then it is known that ε_i approaches a zero mean Gaussian random variable whose variance decreases at a rate of $1/K$ by the central limit theorem [130].

We briefly highlight the fact that the only major assumption made in (4.9) is that we can generate independent noisy measurements of the closed-loop performance functions. We do not require any specific knowledge of the dynamics, or uncertainty distribution, which makes the proposed hardware-software co-design approach broadly applicable. However, the more knowledge that we can exploit in the specification of the expert policy (4.5), the better the choice of the $\xi \in \Xi$ space, which can simplify the process of solving (4.9).

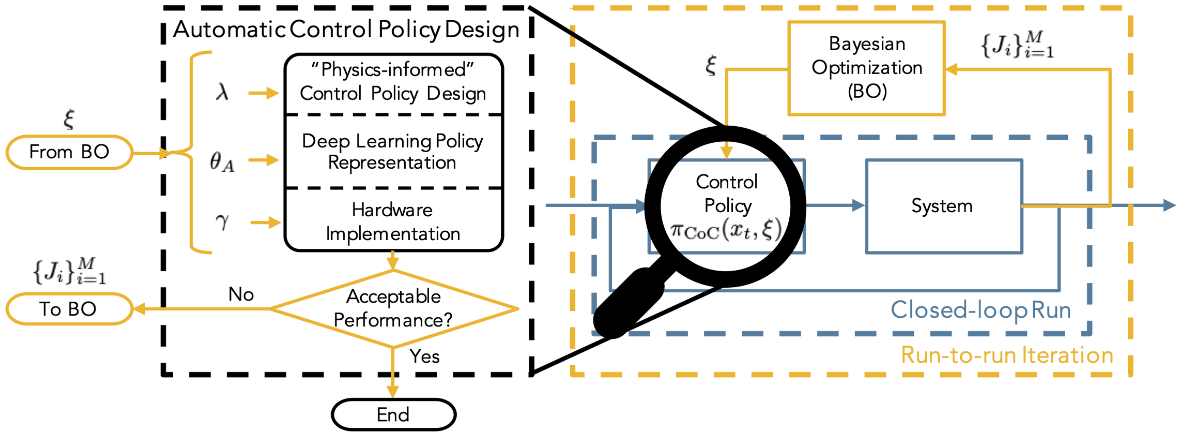


Figure 4.3: Diagram of the data-driven optimization framework. The optimization framework consists in (i) an inner learning procedure that represents a templated workflow to design a single CoC policy (black dashed box) and (ii) an outer optimization stage that suggests new CoC designs via closed-loop evaluations (yellow dashed box). The inner learning procedure (i) is similar to Fig. 4.2, but rather than iteratively optimizing between steps as in Fig. 4.2, the outer optimization (ii) allows us to select parameters from each step (λ, θ_A, γ) concurrently.

4.4.1 A Bayesian Optimization Approach to Co-Design

Since $\{J_i\}_{i=1}^M$ are noisy, expensive functions defined over a relatively low-dimensional space $\xi \in \Xi$, Bayesian optimization (BO) is a natural choice of DFO framework for (4.9) since it is specifically designed for such cases. Furthermore, BO has been shown to surpass state-of-the-art performance in real-world controller tuning applications with a variety of policy types [69]. BO falls under the paradigm of *active learning*, meaning it translates the optimization task into an iterative learning task. There are two major components in BO. First, we must construct a *probabilistic surrogate model*, typically a Gaussian process (GP) [90], to provide a posterior distribution $\mathbb{P}\{\boldsymbol{J}|\mathcal{D}_n\}$ over the unknown true vector-valued function values $\boldsymbol{J}(\xi) = (J_1(\xi), \dots, J_M(\xi))$ given a prior dataset $\mathcal{D}_n = \{(\xi_i, \boldsymbol{y}_i)\}_{i=1}^n$. Second, we must define an *acquisition function* $\alpha_n : \Xi \to \mathbb{R}$ that uses the surrogate model to assign a utility value to the future candidate points at which we can evaluate the true function. Thus, for a well-designed α_n, we would like to preferentially sample at a point that produces the highest possible value. The active learning process is then defined by

$$\xi_{n+1} = \arg\max_{\xi \in \Xi} \alpha_n(\xi). \tag{4.10}$$

Since the surrogate approximation of $\boldsymbol{J}$ is expected to be much cheaper than the true function, we can (approximately) solve (4.10) using established optimization algorithms.

Remark 2 *While GPs are the standard surrogate model-of-choice for BO, they are known to scale poorly with the number of data points D, requiring $O(D^3)$ floating point operations for exact inference, and higher number of data points may be required to obtain representative models for higher dimensional problems. However, there are recent advances that reduce the computational cost at the cost of accuracy (e.g., [131,132]). Additional ways to address large data problems, including using a different type of surrogate model, are an active area of open research [133,134].*

Since we are interested in MOO, we do not have a single best solution and instead would like to provide the control practitioner with an estimate of the set of *Pareto optimal* solutions. A point $\xi \in \Xi$ is considered to be Pareto optimal if improvement in one objective means deteriorating one or more of the others. The so-called Pareto frontier is the set of Pareto optimal points, which is mathematically defined as

$$\mathcal{P}^\star = \{\boldsymbol{J}(\xi) : \nexists \xi' \in \Xi \text{ s.t. } \boldsymbol{J}(\xi') \succ \boldsymbol{J}(\xi)\}, \tag{4.11}$$

where $\boldsymbol{J}(\xi') \succ \boldsymbol{J}(\xi)$ implies the point ξ' dominates the point ξ, which occurs if $J_i(\xi') \geq J_i(\xi)$ for all $i = 1, \dots, M$. To derive α_n, we would like to select points that grow our understanding of $\mathcal{P}^\star$. Following previous work [76,135,136], we use the expected hypervolume improvement (EHVI) acquisition function defined as follows

$$\alpha_n(\xi) = \mathbb{E}\{\text{HV}(\mathcal{P} \cup \{\boldsymbol{J}(\xi)\}, \boldsymbol{r}) - \text{HV}(\mathcal{P}, \boldsymbol{r})\}, \tag{4.12}$$

where $\text{HV}(\mathcal{P}, \boldsymbol{r})$ denotes the hypervolume of a finite approximate Pareto set $\mathcal{P}$ and a reference point $\boldsymbol{r} \in \mathbb{R}^M$ that bounds $\mathcal{P}$ from below. The HV can be computed exactly as the M-dimensional Lebesgue measure

$$\text{HV}(\mathcal{P}, \boldsymbol{r}) = \lambda_M \left(\bigcup_{i=1}^{q} [\boldsymbol{r}, \boldsymbol{y}_i] \right), \tag{4.13}$$

where $\mathcal{P} = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(q)}\}$ is composed of a finite set of q points. EHVI is notoriously difficult to optimize since it has a relatively higher computational cost compared to standard single-objective BO acquisition functions (such as expected improvement) when evaluated using box decomposition. However, as shown in [135], one can more efficiently compute EHVI and its gradients exactly (up to a Monte Carlo integration error) using the inclusion-exclusion principle [137], making the solution of (4.10) tractable.

An illustrative summary of the complete hardware-software co-design framework is provided in Fig. 4.3.

4.4.2 Kernel Selection for Multi-Objective Controller Tuning

The choice of the covariance (or kernel) function in the GP model for $\boldsymbol{J}$ is a critical parameter in the proposed multi-objective BO approach. A particular challenge is the fact that ξ may consist of *ordinal* and *categorical* variables. Ordinal variables are those with some type of natural ordering such as continuous (e.g., weight parameters in the control policy) and integer variables (e.g., the number of nodes/layers in the DNN policy). Categorical variables, on the other hand, are best described as a collection of unordered categories such as the choice of parallelization option in the code generation tool. The standard way for dealing with categorical variables in GP models is to apply one-hot encoding (i.e., converts a c-category variable into c new binary variables). The main challenge with one-hot encoding is that it can lead to a large increase in the dimensionality of the search space as well as complicate the acquisition optimization process. As such, we pursue a mixed kernel function approach [123] that combines separate kernels for the ordinal and categorical variables. For every element of $\boldsymbol{J}$, we focus on independent kernels of the following form

$$k(\xi, \xi') = k^1_{\text{cat}}(\xi_{\text{cat}}, \xi'_{\text{cat}}) k^1_{\text{ord}}(\xi_{\text{ord}}, \xi'_{\text{ord}}) + k^2_{\text{cat}}(\xi_{\text{cat}}, \xi'_{\text{cat}}) + k^2_{\text{ord}}(\xi_{\text{ord}}, \xi'_{\text{ord}}), \tag{4.14}$$

where ξ_{cat} and ξ_{ord} denote the categorical and ordinal components of ξ, respectively, and k^1_{cat}, k^2_{cat}, k^1_{ord}, and k^2_{ord} are kernels associated with the categorical and ordinal variables, each with their own set of hyperparameters. The k^1_{cat} and k^1_{ord} kernels are associated with a product in (4.14), so can capture the joint impact of both types of variables. The k^2_{cat} and k^2_{ord} kernels are associated with a summation in (4.14), so focus on independent impacts of each type of variable.

Based on the idea of Hamming distances, the categorical kernel is selected as follows

$$k^i_{\text{cat}}(x, y) = \nu_i \exp(-d(x, y)/l_i), \quad i \in \{1, 2\}, \tag{4.15}$$

where $d(x, y)$ is meant to represent the distance between categories (equal to 0 if $x = y$ and 1 otherwise) and ν_i is a variance hyperparameter related to the magnitude of the function and l_i is a lengthscale hyperparameter related to how fast the function can vary with distance. For the ordinal variables, we focused on a Matérn-5/2 kernel that have similar variance and lengthscale hyperparameters, though this choice could easily be replaced with any other established kernel (see, e.g., [90] for details on kernel choices and properties).

4.5 Simulations of Hardware Challenges with an Illustrative Example

As a first look into the complexity of the embedded control problem, we examine the benefits offered by using deep learning as a bridge between hardware and software. A major consideration in embedded control is whether or not the proposed control policy can be implemented on hardware, which is related to the hardware constraint (4.1f), and can be a factor of the online computational complexity of the control policy. This section examines the online evaluation of various control policies to illustrate the need for hardware considerations during the software policy development.

Consider a system of masses attached by springs as given in [138]. In general, for a system of m masses, the system can be described by a linear state space model with the m positions and m velocities of each mass as the states and applied force to $m-1$ masses as the inputs. Due to hardware considerations,[2] we use a simplified two-mass system. A first-order hold discrete-time model with sampling time of 0.5 seconds is derived from the continuous-time dynamics such that the plant is of the form

$$x_{k+1} = Ax_k + Bu_k + Gw_k, \tag{4.16}$$

for fixed (A, B, G) matrices given in [138]. We assume the disturbances w_k are independent uniform random variables acting on each mass with each element between -0.5 and 0.5. We can then derive a nominal MPC policy whose goal is to keep the system at rest starting from an initial condition of $x_0 = 0$ by solving the minimization problem

$$\begin{aligned}
\min_{\{u_{t|k}\}_{t=0}^{N_p-1}} \quad & \sum_{k=0}^{N_p-1} x_{t|k}^\top Q x_{t|k} + u_{t|k}^\top R u_{t|k}, && (4.17)\\
\text{s.t.} \quad & x_{t+1|k} = Ax_{t|k} + Bu_{t|k}, \quad t = 0, \dots, N_p - 1, \\
& x_{\min} \le x_{t|k} \le x_{\max}, \qquad t = 1, \dots, N_p, \\
& u_{\min} \le u_{t|k} \le x_{\max}, \qquad t = 0, \dots, N_p - 1,
\end{aligned}$$

[2]The exact system described in [138] (12-state, 3-input) is computationally challenging for standard explicit MPC tools, taking more than five days to generate the solution offline and using more than $30,000$ look-up values.

where $Q = \text{diag}(1, 1, 1, 1)$ and $R = 1$ are the state and input weight matrices, respectively, and the state and input bounds are given by $x_{\min} = [-4, -4, -4, -4]^\top$, $x_{\max} = [4, 4, 4, 4]^\top$, $u_{\min} = -0.5$ and $u_{\max} = 0.5$.

We evaluate the embeddability of three control policies: an implicit MPC, an explicit MPC (EMPC), and a DNN approximation to MPC. Implicit MPC refers to the online solution to optimization problem (4.17). We use CasADi to formulate the problem as a quadratic program and solve using QRQP [139]. EMPC subdivides the state-space into polytopic regions with precomputed controller gains. EMPC can typically be readily embedded on hardware since the control law is reduced to a look-up table [140]. The optimization problem is created with the MPT-3 toolbox [141] by using the `MPCController()` function, which is the equivalent to the implicit MPC created using CasADi. An explicit control policy is then created by calling the `toExplicit()` function, which solves a multiparametric programming problem to determine the piecewise affine (PWA) function that replaces the online optimization problem. The evaluation of the explicit control policy is performed by conducting sequential search on the returned set of gains for the PWA representation. Note that when the EMPC is exported to C-code, the search is conducted via more efficient binary search trees [141]. Finally, the DNN-based approximate MPC is trained using data generated offline by solving the implicit MPC law (4.17). The DNN is trained following the procedure described in Section 4.3.3 using $n_s = 5,000$ samples with MATLAB's `feedforwardnet` function. Note that the choice of n_s and the quality of the training data must be done with respect to the desired accuracy and robustness, where the choice of $n_s = 5,000$ was done empirically by performing some initial training, validation, and testing results on randomly selected architectures.

Each method was evaluated across $1,000$ replicate simulations with 100 time steps each resulting in $100,000$ total time steps. Disturbance realizations were consistent for each controller, but varied in each replicate simulation. To mimic embedded implementation, each control policy was converted to C-code and compiled into `MEX` functions. Each of CasADi, MPT-3, and MATLAB has a routine to convert the m-code into C-code. Once C-code is generated, each program is compiled into a `MEX` function. A `MEX` function is a MATLAB function that calls a C program, acting as an interface between a high-level programming language (i.e., MATLAB) and a low-level one (i.e., C). Settings and the procedures used in the compilation process are detailed in the Section 4.8.[3]

Table 4.1 compares three metrics of interest for each control policy: closed-loop performance, computational time, and memory requirements. The closed-loop performance is given as the average stage cost computed over the $100,000$ steps of the system. Additionally, Fig. 4.4 depicts the distributions of the stage cost for each controller. The computational time is given as the average time taken to compute an input using a compiled C-program (via `MEX` function) on a general purpose laptop CPU (2.3 GHz 8-Core Intel i7 processor).[4] The memory requirements are given in two forms: one related to the general storage of

[3]All code and additional documentation (including those for the results and discussion of the next section) may be found at `https://github.com/Mesbah-Lab-UCB/HW-SW_CoDesign4CoC`.

[4]On a less powerful device and/or on specialized hardware, the results in Table 4.1 would show even

Table 4.1: Comparison of control policies in the illustrative example on general purpose CPU: MPC, standard explicit MPC (EMPC), and neural network approximation of MPC (DNN)

	Stage Cost	Computation Time (ms)			Memory (kB)	
	Average	Average	Max	Min	File Size	Max Heap
MPC	4.77 ± 5.74	0.55 ± 0.11	1.91	0.37	242.5	5.0
EMPC	4.81 ± 5.92	1.40 ± 0.13	2.76	1.32	5600.0	6.8
DNN	4.76 ± 5.67	0.01 ± 0.01	0.38	0.01	18.0	5.0

the program (executable file size) and one related to the random access memory (RAM) required during computation (maximum heap utilization). The closed-loop performance between the three controllers are similar, if not the same. However, when considering hardware constraints/utilization, standard EMPC takes the longest computation time and greatest memory consumption. In general, standard EMPC is ill-suited for this problem since even the implicit solution can outperform EMPC. Regardless, the DNN can offer an order-of-magnitude improvement in computation speed and memory utilization as compared to the MPC. The gap between MPC implementations and DNN implementations can be significantly widened in specialized contexts, e.g., in cold plasma bioprocessing/medicine, where the dynamics are nonlinear and the control context requires safety considerations.

4.6 Results of Control-on-a-Chip for Cold Atmospheric Plasma Jets

To illustrate the complete proposed design framework illustrated in Fig. 4.3, we investigate the CoC design for cold atmospheric plasma (CAP) devices. CAPs have recently found promising use in a variety of applications, including (bio)materials processing [142] and plasma medicine [14–16]. CAPs, a low-temperature (partially) ionized gas, can be generated by applying electric fields to a noble gas, typically argon or helium. The synergistic effects of CAPs, including the generation of reactive chemical species and ions, ultraviolet radiation, low-level electric fields, and thermal effects, are posited to induce therapeutic and practical outcomes [143, 144]. CAP devices, such as atmospheric pressure plasma jets (CAPJs) [80], can facilitate direct CAP treatments by providing a portable, point-of-use solution to deliver plasma effects in a directed manner. However, CAPJs pose unique challenges in control and rely on cutting-edge control formulations [2, 145], most of which have no unified embedded

larger differences.

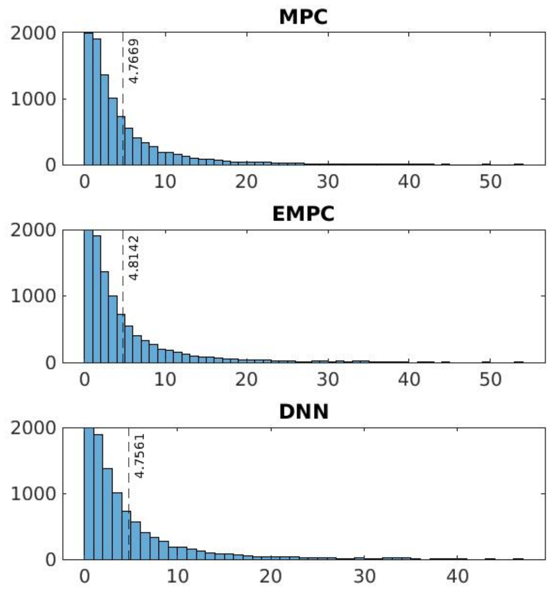


Figure 4.4: Distributions of the stage cost for three control policies used in the illustrative example: implicit MPC, standard explicit MPC (EMPC), and DNN approximation to MPC (DNN).

techniques. This section will cover some key additions and/or deviations from the description of the CAPJ system in Section 2.2 and demonstrate a CoC design study for the CAPJ system.

4.6.1 Resource-limited Hardware Integration with CAPJ

In this chapter, we added a field programmable gate array (FPGA) to the CAPJ system described in Section 2.2 to act as the low-resource hardware for embedded control systems. Data acquisition was implemented and managed via USB connection to a standard CPU using Python. DNN-based CoC policies were implemented on a FPGA (the programmable logic side of a Zybo Z7, XC7Z020-1CLG400C). The programming files for the FPGA were generated automatically using MathWorks HDL Coder (included with MATLAB R2021a) and Xilinx Vivado 2020.1. FPGA-in-the-loop simulations and experiments were facilitated by MATLAB on a standard laptop CPU, rather than the Zybo Z7's on-board processor.[5]

[5]Note that the communication time between devices can play a role in effective hardware implementation. Further investigation into using a system-on-chip architecture provided by the Zybo Z7 is left for future work.

4.6.2 Modeling for Control of CAPJs

A model of the CAPJ is used as described in Section 2.2.2, and the key equations are revisited here. The dynamics of the CAPJ is described by a data-driven LTI model

$$x_{k+1} = Ax_k + Bu_k, \tag{2.1a revisited}$$
$$y_k = Cx_k + Du_k, \tag{2.1b revisited}$$

and the thermal effect of the CAP is given by

$$\mathrm{CEM}_{k+1} = \mathrm{CEM}_k + K^{(T_{\mathrm{ref}} - T_{s,k})}\delta t. \tag{2.3 revisited}$$

4.6.3 Scenario-based Model Predictive Control

Since we have no way to remove thermal effects once delivered, we need to be cautious in the face of uncertainty. Therefore, we select scenario-based MPC [146], or sMPC for short, to provide a controller that is robust to uncertainty especially in the presence of safety-critical constraints. Specifically, sMPC assumes that the system uncertainty can take on finite number of s scenarios at every time step. Whenever the uncertainty is time-varying in the sense that it can take on new values at every time step, the system evolution can be represented by a *scenario tree* of $S = s^N$ unique combinations of uncertainty values where N denotes the prediction horizon [147]. The sMPC policy then solves the following minimization problem at every time step

$$\min_{u_{i,j|k}} \sum_{j=1}^{S} \omega_j \left[\sum_{i=0}^{N_p-1} V(x_{i,j|k}, u_{i,j|k}) + V_f(x_{N_p,j|k}) \right], \tag{4.19a}$$
$$\text{s.t. } x_{i+1,j|k} = Ax_{i,j|k} + Bu_{i,j|k} + w_{i,j|k}, \tag{4.19b}$$
$$(x_{i,j|k}, u_{i,j|k}) \in \mathcal{X} \times \mathcal{U}, \tag{4.19c}$$
$$\textstyle\sum_{j=1}^{S} \tilde{E}_j U_{j|k} = 0, \tag{4.19d}$$
$$x_{0,j|k} = x_k, \tag{4.19e}$$
$$\forall i \in \{0, \ldots, N_p - 1\},\ \forall j \in \{1, \ldots, S\},$$

where the subscript $(\cdot)_{i,j|k}$ denotes the j-th scenario predicted i steps ahead of the current time k, ω_j is the probability of occurrence of the j-th disturbance sequence $W_{j|k} = (w_{0,j|k}, \ldots, w_{N_p-1,j|k})$, $\mathcal{X}$ and $\mathcal{U}$ are the state and input constraints, respectively, that must hold for all uncertainty realizations, V and V_f are the stage and terminal cost functions, respectively, and (4.19d) enforces the *non-anticipativity constraints*, which ensure states that branch from the same parent node have the same control input. As shown in [148], these constraints can be written in terms of known matrices $\{\tilde{E}_j\}_{j=1}^{S}$ that impose structure on the vector of control inputs for the j-th scenario, i.e., $U_{j|k} = (u_{0,j|k}, \ldots, u_{N_p-1,j|k})$. To limit the exponential growth in the scenario tree, we use the idea of the robust horizon $N_r < N_p$

Table 4.2: Examples of Design Parameters in the Control-on-a-Chip (CoC) Design Process

CoC Design Step	Examples
Scenario-based MPC (λ)	prediction horizon (N), robust horizon (N_r), back-off parameters, uncertainty bounds ($\hat{w}_{\text{bound}}$)
DNN Approximation (θ_A)	number of layers (L), number of nodes (H), activation function(s) ($\{\alpha_l\}_{l=1}^{L}$), training/optimizer parameters
Hardware Implementation (FPGA) (γ)	numerical representation (e.g., total/fraction length/bit representation), parallelization options

from [147], which stops branching after N_r steps with N_r usually equal to 2 or 3 (often sufficient in practice to achieve constraint satisfaction).

In (4.19), the control objective for the CAPJ system is given by a terminal cost of $V_f(x_{N_p}) = (\text{CEM}_{\text{sp}} - \text{CEM}_{N_p|k})^2$ where CEM_{sp} is the desired CEM value. We use a prediction horizon of $N_p = 5$ and a robust horizon of $N_r = 2$. We consider a set of 3 scenarios at each time step, mainly $\{(-\hat{w}_{\text{bound}}, -\hat{w}_{\text{bound}}), (0,0), (\hat{w}_{\text{bound}}, \hat{w}_{\text{bound}})\}$, which correspond to low, middle, and high values for the uncertainty for a tuning parameter $\hat{w}_{\text{bound}}$. The inputs are constrained by the hardware to satisfy $P \in [1.5, 5.0]$ W and $q \in [1.5, 5.0]$ s.l.m. The outputs are constrained by the following limits $T_s \in [25, 45]$°C and $I \in [0, 80]$ intensity units. The optimization problem (4.19) was formulated using CasADi [139] and solved with IPOPT [102] in a receding-horizon fashion.

Although sMPC provides an effective robust control strategy for the CAPJ, it poses a significant challenge for CoC design since, to the best of the authors' knowledge, there are no known off-the-shelf embedded implementations of (4.19). As such, sMPC provides a useful expert control policy that can be imitated by a DNN on embedded hardware.

4.6.4 Hardware-in-the-loop Simulations

Now, our goal is to embed the sMPC policy defined by (4.19) on specialized hardware, mainly an FPGA (i.e., the programmable logic side of the Zybo Z7). We can select CoC design parameters at each stage of the proposed framework – we list several possible parameters in Table 4.2. We selected the following closed-loop performance metrics to define the MOO problem (4.9) that provides the basis for selecting the complete set of software and hardware design parameters ξ

$$J_1(\xi) = \mathbb{E}\left\{\sum_{k=0}^{T} \left(\text{CEM}_{\text{sp}} - \text{CEM}_k(\xi, W))^2\right)\right\}, \tag{4.20a}$$

$$J_2(\xi) = \mathbb{E}\left\{\sum_{k=0}^{T} \left([T_{s,k}(\xi, W) - T_{\max}]^+)^2\right)\right\}, \tag{4.20b}$$

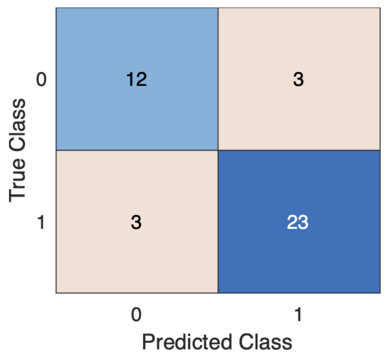


Figure 4.5: Confusion matrix of the hardware feasibility classifier. Recall that 0 represents an infeasible control-on-a-chip design, while 1 indicates a feasible design.

where $W = (w_0, \ldots, w_{T-1})$ is the set of random uncertainty values over the horizon T, $[a]^+ = \max\{a, 0\}$, and the expectation is taken over W. Here, J_1 represents deviation from the desired setpoint of $\mathrm{CEM}_{\mathrm{sp}} = 1.5$ min and J_2 is a temperature constraint violation metric given $T_{\max} = 45°\mathrm{C}$.

We select the following CoC design parameters and their bounds: maximum possible uncertainty realization $\hat{w}_{1,\mathrm{bound}} \in [0, 10]$, number of nodes per hidden layer in the DNN $H \in [2, 10]$, and the fixed-point word length when generating code $wl \in [10, 32]$.

Hardware Feasibility Classifier Results

As mentioned in Section 4.3.2, the constraint function $g(\theta_A, \gamma)$ can be learned prior to any optimization routine. As such, we pre-select a range of H and wl to learn a binary classifier $\tilde{g}(H, wl) \rightarrow \{0, 1\}$. Data $\mathcal{D}_g = \{(H, wl), g(H, wl)\}$ were generated by creating DNNs with randomly initialized weights and passing them through the code generation process $\mathcal{CG}$. Feasibility was recorded with each combination of parameters. Using $\mathcal{D}_g$, a neural-network-based classifier was trained using MATLAB's `fitcnet` function with an 80/20 training/test split.

Remark 3 *While the generation of training data for the hardware feasibility classifier is performed independently of the CoC design optimization, the process of generating the data itself can be costly due to the time required to run code generation step (i.e., the FPGA synthesis). To reduce the cost of gathering training data, optimization methods like BO can be used to determine the boundaries of feasible/infeasible points instead of searching over the entire parameter space of* Θ_A *and* Γ*.*

Fig. 4.5 shows the confusion matrix of the resulting hardware classifier learned from $\mathcal{D}_n$. The confusion matrix shows a reliable estimation of how well we can predict the feasibility of a particular hardware design. As shown in Fig. 4.5, the classifier accurately predicts with an

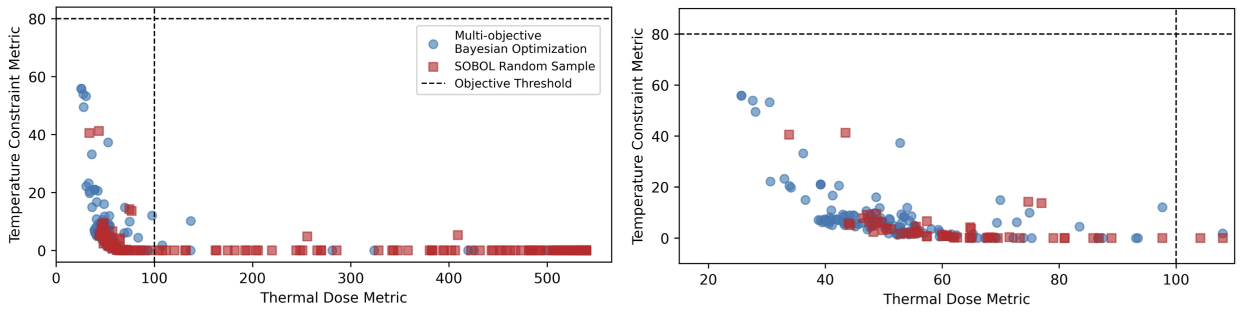


Figure 4.6: Observed closed-loop performance metrics of plasma treatments during five replicates each of multi-objective Bayesian optimization (MOBO) and random search via SOBOL sampling. Blue circles indicate the metrics observed during MOBO; red squares indicate the metrics observed during random search. Dashed black lines indicate "constraints" on the closed-loop performance metrics that are used to guide parameter suggestions to the region-of-interest. The left figure shows all data encountered in all optimization routines, while the right figure illustrates a zoomed-in version (truncating the upper x-axis value at 120). Note that random search has no notion of the objective threshold since no surrogate model is created based on previously observed data. As such, random search explores significantly more designs outside of the region-of-interest.

85% test accuracy. This hardware classifier can then be used during an optimization routine to estimate which combination of H and wl are feasible. The result is that we effectively reduce the search space prior to performing full CoC design runs.

Multi-objective Framework Results

Five replicates of MOBO are used to iteratively find the Pareto frontier according to the two closed-loop performance metrics using a total of 50 iterations. Closed-loop FPGA-in-the-loop simulations of the CAPJ are performed in MATLAB. DNNs are trained with `feedforwardnet` and $n_s = 2,000$. Again, the choice of n_s was done empirically. However, a nice feature of our CoC framework is that the closed-loop validation process is able to "catch" if poor performance is consistently achieved for a variety of design parameters ξ, which, in turn, may be indicative of a poor training data set. Since the CoC optimization is performed offline (up until the real-system query), one can always increase the size of n_s at the cost of longer data generation and training time. Results from the closed-loop simulations are passed into the MOBO framework implemented in Python using Ax [123]. MOBO is compared to random search (RS) using a quasi-random Sobol sampling strategy [149], which is a common benchmark for derivative-free optimization methods. Each strategy was initialized with one known, successful design and one Sobol-selected design.

Fig. 4.6 illustrates the observed closed-loop performance metrics over the five replicates of

MOBO and RS. In Fig. 4.6, the blue circles indicate data points gathered from MOBO, while red squares indicate data points gathered from RS. Dashed black lines indicate "objective thresholds," which are used in Ax to "constrain" the search space to produce values within a desired region-of-interest. Objective thresholds are chosen such that the closed-loop performance metrics are representative of valuable or practical control policies. In other words, in the case of the thermal dose metric, if the metric exceeds 100 ($\approx \text{CEM}_{sp} \times 70$ time steps), then this would mean that the suggested CoC policy will take longer than the maximum treatment time allowed and/or is incapable of achieving the desired thermal dose. In the case of the temperature constraint metric, if the metric exceeds 80 ($\approx 1°\text{C}\times 80$ time steps or $10°\text{C}\times 8$ time steps), then the policy is considered too dangerous as it violates the constraint too often or at too high magnitude. Objective thresholds can be chosen since MOBO produces a surrogate model that relates the design parameters to the performance metrics. The posterior model can be used to estimate parameters that will produce performance metrics that are likely to be in the defined region-of-interest. Since the surrogate model in MOBO is probabilistic in nature, CoC design parameters suggested by MOBO may still fall outside of the region-of-interest in the initial iterations of MOBO. Note that the RS has no notion of the objective threshold since no surrogate model is created based on the previously-observed data. As such, RS explores significantly more designs outside of the region-of-interest due to a less constricted parameter space. The left subfigure shows all data encountered, while the right shows a zoomed-in version with a truncated x-axis, and it illustrates how RS encounters significantly more points with a thermal dose metric greater than 100.

Furthermore, Fig. 4.7 shows the HV evolution over the 50 iterations of each method (MOBO in blue and RS in red). The solid lines indicate the mean over five replicates whereas the shaded region indicates one standard error. Recall that the HV is a measure of the quality of the Pareto frontier. As seen in Fig. 4.7, the two methods begin in a similar fashion at suboptimal Pareto frontier estimations over the first two iterations, but MOBO quickly diverges within the immediate next few iterations. MOBO's increase in HV value in few iterations is due to its ability to intelligently explore the design space in search of the optimal trade-off between the two performance metrics. Meanwhile, RS is a naïve approach that explores many options that are not expected to improve the HV, i.e., designs are selected with no knowledge of the outcome. As such, it takes RS many more iterations to reach a HV close to that of a "converged" MOBO.

Finally, Fig. 4.8 shows closed-loop trajectories of the states related to J_1 and J_2 (CEM and T, respectively). We use Fig. 4.8 to illustrate the performances of various CoC designs that are encountered at several snapshots of MOBO. Using Fig. 4.8, we can see the variety of control policies that are encountered, and how the observed Pareto frontier evolves over time. From top to bottom, we show snapshots of one replicate of MOBO at iterations 5, 15, and 25. From left to right, the subfigures in the left column are the closed-loop trajectories of CEM for selected designs; the subfigures in the middle column are the closed-loop trajectories of surface temperature for selected designs; and the subfigures in the right column are the observed metrics from the designs encountered up until that iteration. We selected designs that a practitioner may select based on the application needs. A "utopia"

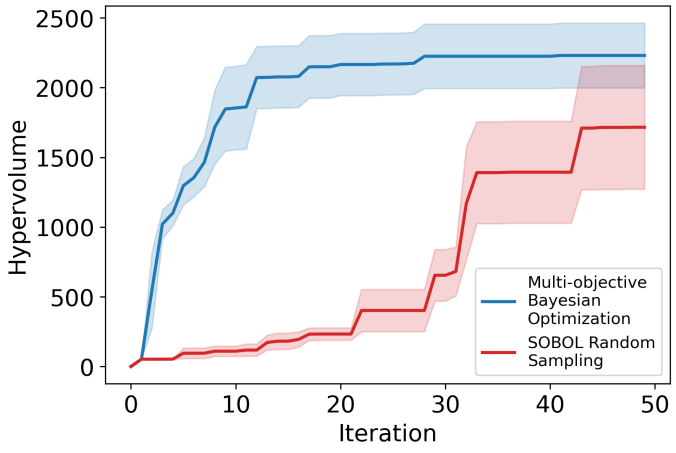


Figure 4.7: Hypervolume improvement of five replicates each of multi-objective Bayesian optimization (MOBO, blue) and random search via SOBOL sampling (red) for the control-on-a-chip design for atmospheric pressure plasma jets. Solid lines indicate the mean hypervolume and the shaded regions indicate one standard error. MOBO on average reaches a higher hypervolume overall and earlier than random sampling, which indicates that a meaningful Pareto frontier is realized in fewer iterations of MOBO than of random sampling.

design is determined on the basis of the lowest combination (scaled sum) of performance metric values,

$$\Psi_u = \hat{J}_1 + \hat{J}_2,$$

where $\hat{J}_i$ are scaled values of J_i. Bounds of the scaling are fixed to $[20, 100]$ for J_1 and $[0, 60]$ for J_2. A "control performance preferred" design is determined on the basis of a weighted combination of J_1 and J_2,

$$\Psi_p = \tau \hat{J}_1 + \frac{1}{\tau}\hat{J}_2,$$

where $\tau = 3$. A "constraint satisfaction preferred" design is determined similarly, but with the weights switched, i.e.,

$$\Psi_s = \frac{1}{\tau}\hat{J}_1 + \tau \hat{J}_2.$$

Designs are selected in this manner to avoid overly extreme controller designs. Utopia designs are denoted by the color green (and solid lines in the trajectory plots); control performance preferred designs are denoted by the color orange (and dotted-dashed lines); constraint satisfaction preferred designs are denoted by the color brown. Additionally, if any selected designs were the first artificial point, then they are colored purple. At iteration 5, MOBO

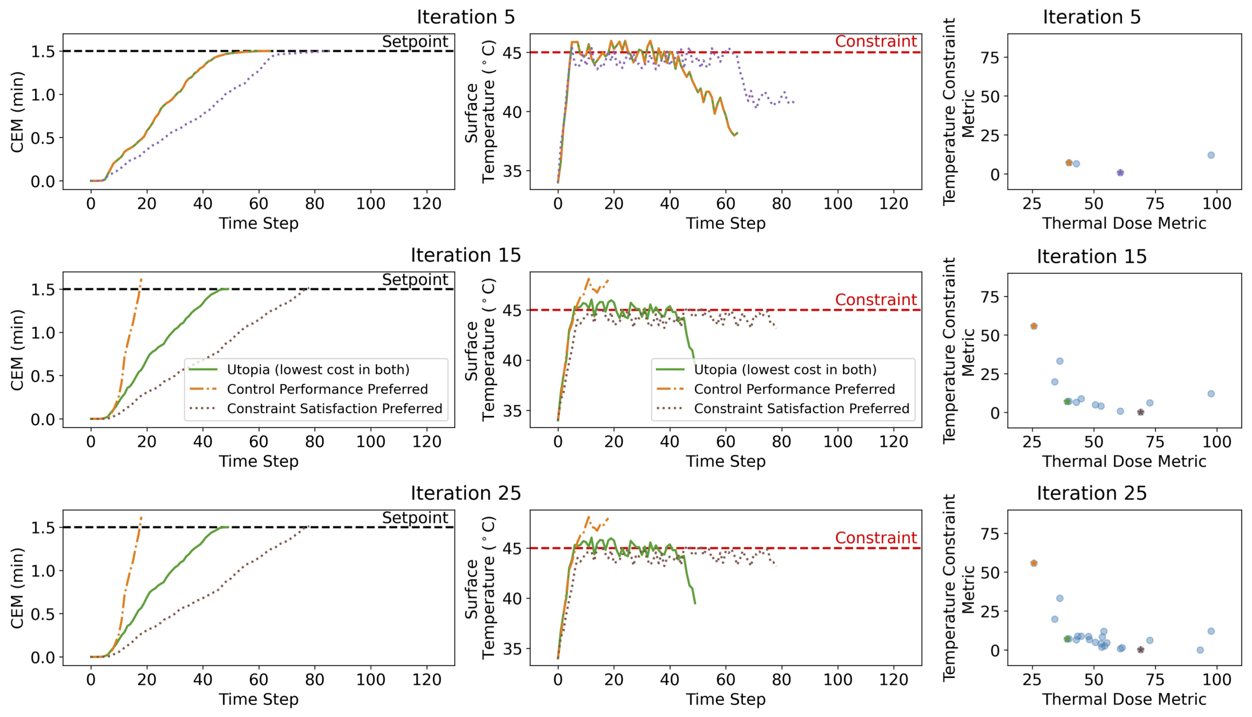


Figure 4.8: Closed-loop trajectories of CEM (left column) and surface temperature (middle column) at snapshots (at Iterations 5, 15, and 25) of one replicate of multi-objective Bayesian optimization. Selected designs (colored stars) were determined from the observed data (blue circles) in the metric space (right column). The selected designs correspond to designs that an engineer may select based on the needs of a particular application. The "utopia" point/design (solid green) is selected on the basis of lowest combination (scaled summation) of the metric values. The "control performance preferred" point (dot-dashed orange) is selected on the basis of a weighted combination of the metric values where the thermal dose metric is weighted three times more than the temperature constraint metric. The "constraint satisfaction preferred" point (dotted brown) is selected on the basis of a weighted combination of the metric values where the temperature constraint metric is weighted three times more than the thermal dose metric. In the CEM figures, the dashed black line represents the desired thermal dose. In the surface temperature figures, the dashed red line represents the constraint.

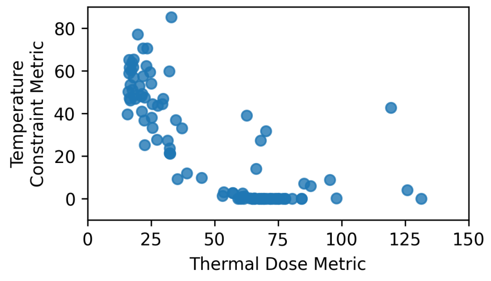


Figure 4.9: Observed performance metrics during three replicates of multi-objective Bayesian optimization for control-on-a-chip design based on design parameters chosen for the experimental case study. Modifying different design parameters still shows a similar trade-off between the two closed-loop performance metrics.

has selected designs that improve the Pareto frontier such that a utopia design and control performance preferred design are different from the first design. The utopia point provides an indication of design with the most even trade-off between the different metrics and can also partially indicate the quality of the Pareto frontier. At iteration 15, more designs have been explored such that new designs for the more extreme designs can be observed. At iteration 25, several more designs have been explored, but had little value to finding the Pareto optimal points, and as a result, had no significant changes between iteration 15.

4.6.5 Real-time Experimental Verification

Finally, to show utility in the real system, we selected three control policy designs from the Pareto frontier generated in simulation. Note, for ease of testing, we chose to modify H, L, wl, and *loop* where *loop* is a categorical option in the code generation process that determines whether or not to parallelize certain matrix computations. We used MOBO offline to search for optimal CoC designs and transfer the designs to real-time experiments. In principle, these optimal design parameters obtained offline are sufficient for the experiments as they primarily describe the capability of the CoC policy in (i) accurately representing the sMPC law and (ii) being feasibly implemented on the hardware device. Fig. 4.9 shows the observed data from three replicates of MOBO for the new set of design parameters. Fig. 4.9 shows a similar trade-off as in Fig. 4.6 even with different design parameters. Real-time experiments using the CoC design parameters to create embedded control policies for the CAPJ were performed in triplicates. Table 4.3 describes the controller parameters used and the closed-loop metrics obtained on the CAPJ testbed. Due to the significant plant-model mismatch between the linear models and the true system, the original sMPC was tuned based on a newly identified LTI model on the day of experiments to achieve appropriate control performance before

Table 4.3: Configurations and Closed-loop Metrics for Real-time Control Experiments with the APPJ Testbed. Metrics are reported as the mean $\pm$ the standard error of 5 replicates.

		Control Policy Parameters				Closed-loop Performance Metrics	
	Description	Hidden Nodes	Hidden Layers	Fixed Point Word Length	Loop Unrolling	Thermal Dose	Temperature Constraint
(i)	Control Performance Preferred	5	2	13	UnrollLoops	35.66 ± 0.59	2.12 ± 0.43
(ii)	Constraint Satisfaction Preferred	8	2	18	LoopNone	39.53 ± 4.19	1.07 ± 0.40
(iii)	Utopia	5	3	20	LoopNone	33.67 ± 2.03	1.78 ± 0.39

testing CoC designs.[6] From Table 4.3, we can see several trade-offs between control performance, hardware utilization, and constraint satisfaction. A performance-dominated CoC design (i.e., Configuration (i)) is typically comprised of a small-scale DNN that can achieve the desired CEM dose quickly at the expense of more constraint violations. A constraint-dominated CoC design (i.e., Configuration (ii)) is typically comprised of a larger width DNN with larger fixed point word length. This often leads to a more representative DNN that has fewer constraint violations. Finally, a mixed CoC design (i.e., Configuration (iii)) can offer a balance of the two extremes.

4.7 Conclusions and Future Work

This chapter presented an end-to-end control-on-a-chip (CoC) design framework for the implementation of arbitrary control policies on arbitrary hardware. We argued for deep learning as a unifying template that connects the hardware and software aspects of embedded control design. Furthermore, we presented a Bayesian optimization framework for CoC design that can account for the multi-objective nature of the control design problem, categorical design spaces, and minimal interactions with the expensive design process. We demonstrated the proposed CoC design framework for cold atmospheric plasma processing of biomaterials in closed-loop simulation studies and real-time experiments. The framework was able to efficiently and systematically determine trade-offs in the CoC design process, resulting in adequate estimation of the Pareto frontier in only a few design iterations. Future work may

[6]New system parameters are provided in Section 4.8.1.

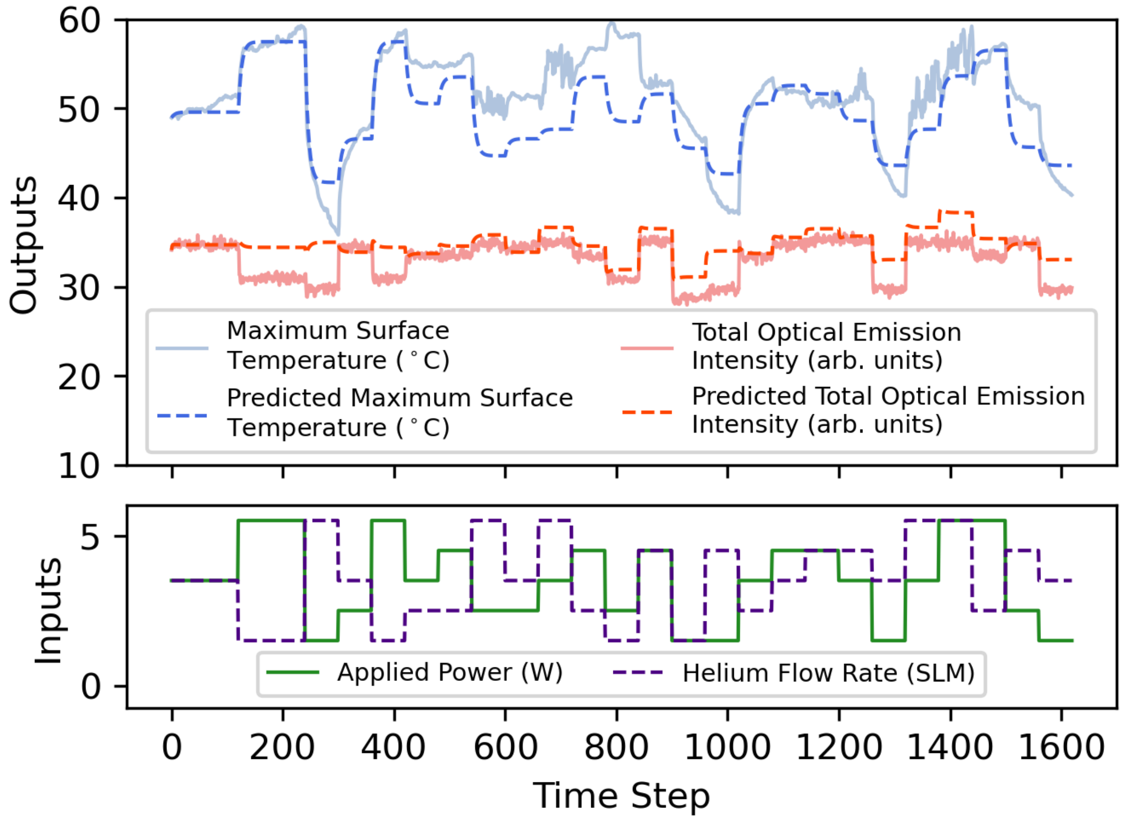


Figure 4.10: Comparison of measured APPJ outputs and the model used in sMPC (4.23). A new model was learned from new experimental data, since the configuration of the APPJ had changed since the data collection for (4.21) and (4.22).

focus on extending the framework to online adaptation of control policies and guaranteeing the robustness (i.e., safety) of the resulting controllers.

4.8 Additional Information

4.8.1 Subspace Identification for the APPJ Testbed

As indicated in the main text, we used subspace identification [86] to identify discrete-time state-space models from open-loop data of the APPJ testbed. For the closed-loop simulations described in Section 4.6, we used two sets of identified matrices to simulate plant-model mismatch. The following matrices were used in defining the control model $\hat{f}(x, u)$

$$A = \begin{bmatrix} 0.903 & 0.018 \\ 0.132 & 0.243 \end{bmatrix}, \quad B = \begin{bmatrix} 0.581 & -0.220 \\ 2.674 & -1.131 \end{bmatrix}. \tag{4.21}$$

The following matrices were used in defining the plant model (2.2)

$$A = \begin{bmatrix} 0.888 & 0.055 \\ 0.094 & 0.283 \end{bmatrix}, \quad B = \begin{bmatrix} 0.503 & -0.174 \\ 2.764 & -1.037 \end{bmatrix}. \tag{4.22}$$

In Section 4.6.5, a new state-space model was identified for sMPC. The matrices for $\hat{f}(x, u)$ are given as follows

$$A = \begin{bmatrix} 0.845 & 0.016 \\ -0.060 & 0.358 \end{bmatrix}, \quad B = \begin{bmatrix} 0.450 & -0.164 \\ 0.713 & 0.567 \end{bmatrix}. \tag{4.23}$$

Furthermore, the nominal operating conditions were given as $[P^s, q^s]^\top = [3.5 \text{ W}, \ 3.5 \text{ s.l.m.}]^\top$ and $[T^s, I^s]^\top = [49.6°\text{C}, \ 29.7 \text{ arb. units}]^\top$. Fig. 4.10 (b) shows a comparison between the open-loop data collected (via multi-step tests) and the model learned from the open-loop data.

4.8.2 Compilation Settings for Generating Embedded Control Policies

This section describes the settings used in generating code for hardware-based control policies. Most tools were used with their default settings with the exception of those detailed in this section.

CasADi Code Generation

As mentioned in the main text, CasADi provides a `generate()` function to create a C-code representation of the optimization problem defined with CasADi's symbolic syntax. The optimization problem (4.17) from the illustrative example in Section 4.5 is formulated using CasADi's symbolic syntax and used to generate a functional form of the control law

$$u_t^\star = \pi_{mpc}(x_t), \tag{4.24}$$

where a call to π_{mpc} solves the implicit optimization problem defined by (4.17). The variable in which π_{mpc} is stored is used with the `generate(filename, opts)` function with these additional options (defined in as the structure `opts`):

- `'main': true`
- `'mex': true`

- `'with_header': true`.

Then, the C-code generated under the filename variable `filename` is converted to a `MEX` function for use as a `MATLAB` function using the following command (within `MATLAB` Command Window): `mex filename -largeArrayDims -outdir codegen/mex/mpc/ -output mpc_mex`.

MPT-3 Code Generation

As mentioned in the main text, MPT-3 provides several methods to generate C-code for control policies defined using MPT-3 syntax. In this work, we use the method that involves the use of the `MATLAB Coder` toolbox. The procedure to do so is as follows:

1. Define a control policy using the `MPCController` object, which takes `system` and `horizon` as arguments, where `system` is the linear model to be used in the dynamics constraints and `horizon` is the prediction horizon of the optimal control problem.

   ```
   c = MPCController(system, horizon);
   ```

2. Convert the control policy into an explicit control policy using the `toExplicit()` function.

   ```
   ec = c.toExplicit();
   ```

3. Convert the explicit control policy to a functional representation of the control law.

   ```
   ec.toMatlab(mfile, 'primal', 'obj');
   ```

   where the `mfile` is an m-code representation of the control law

   $$u_t^\star = \pi_{empc}(x_t), \tag{4.25}$$

   where the evaluation of π_{empc} searches for the appropriate gain from the PWA function that defines the explicit control policy.

4. Use `MATLAB Coder` to create a `MEX` function.

   ```
   coder mfile -args {zeros(nx, 1)};
   ```

   where `nx` is the number of states.

DNN Code Generation

The DNN can be converted to C-code through a similar method as the final step of the MPT-3 method. The DNN was created using the `feedforwardnet` function, which is a part of the Deep Learning toolbox. In addition, the Deep Learning toolbox provides a function `genFunction` that converts a neural network to a `MATLAB` function, i.e., an m-code representation of the DNN control policy

$$\hat{u}_t^\star = \pi_{dnn}(x_t), \tag{4.26}$$

where the evaluation of π_{dnn} is the forward pass of the learned DNN. The DNN is converted with `genFunction`:
`genFunction(net, mfile, 'MatrixOnly', 'yes');`
where `net` is the DNN object generated after calling `feedforwardnet` and `mfile` is the filename of the m-code function that is generated. Finally, the same call to `MATLAB Coder` from Step 4 of the MPT-3 code generation is called to create the C-code and `MEX` function.

Chapter 5

Towards Personalized Plasma Medicine with Bayesian Optimization Particularities[1]

Precision or personalized medicine is an approach that aims to select medical treatments based on factors that are unique to an individual. In plasma medicine, the difficult-to-elucidate plasma-interface interactions and uncertainty around plasma treatment outcomes pose an additional challenge. Data-driven optimization that can mimic the doctor-patient interaction can be useful to inform decisions made by a physician. This chapter illustrates how two aspects of healthcare: balancing multi-objective outcomes and safe exploration can be addressed by data-driven optimization.

5.1 Introduction

Cold atmospheric plasmas (CAPs) have found promising use in plasma medicine [16]. CAPs, a low-temperature (partially) ionized gas, can be generated by applying an electric field to a noble gas, such as argon or helium, whereby the resulting discharge is directed towards a target surface [80]. The synergistic effects of CAPs, including the generation of reactive chemical species and ions, ultraviolet radiation, low-level electric fields, and thermal effects, can induce therapeutic outcomes [143]. As such, portable CAP devices have shown promise for a variety of point-of-care biomedical applications [150]. Nevertheless, CAPs exhibit multivariable, distributed-parameter, and intrinsically variable dynamics and are often subject to (safety-critical) constraints. Thus, there has been a growing interest in advanced control of biomedical CAP devices using model predictive control (MPC) [151] and learning-based control strategies [65]. Two of the main challenges in MPC of CAPs stem from the need to: (i) handle the fast dynamics and, thus, kilohertz (kHz) sampling rates of CAPs [2], and

[1]This chapter was adapted with permission from the coauthors from [76] and [77].

(ii) adapt MPC policy parameters to account for variable characteristics of the plasma and target surfaces [19].

The notion of learning and adaptation, as well as auto-tuning, of control policies using closed-loop performance data has received increasing attention. Notably, policy-gradient methods [152] have been used as a popular reinforcement learning (RL) approach to guide policy search within continuous control-input spaces, with particular success for MPC policies (e.g., [110]). Due to its use of gradient information, policy-gradient RL is touted as a scalable alternative to the increasingly popular Bayesian optimization (BO) strategy for controller auto-tuning (e.g., [153,154]), but at a cost of lower data-efficiency, especially when initialized poorly. Instead, BO can be a viable alternative for data-efficient policy search, especially when performance data and/or interactions with the real environment are limited. BO is a derivative-free, probabilistically principled method for "global" optimization that can handle a mixture of continuous, discrete, and categorical decision variables [70]. For example, [155] presents an entropy-search BO approach to use *prior* information from a "cheap" simulated environment for sample-efficient policy learning on the actual physical system. Moreover, the multi-objective nature of policy search can be directly accommodated in BO, when there is a need to discover a set of optimal policies due to multiple conflicting objectives [136,156].

For plasma treatment of complex interfaces, it is imperative to adapt control policies to account for the variability among different target surfaces, in addition to the time-varying nature of the plasma and surface characteristics. Moreover, adaptability of the treatment policy is important for personalized plasma medicine, where CAP treatments must be tailored for each individual subject to enhance their therapeutic efficacy without compromising the safety and comfort of patients. However, a key challenge arises from the limited number of treatments/trials that can be performed in a biomedical context, which makes data efficiency a prerequisite for control policy adaptation and BO an attractive option for that data-driven control policy adaptation.

5.2 Multi-objective Neural-Network-based Policy Search

This section presents a strategy for adaptive deep learning-based approximate MPC, towards personalized and point-of-care biomedical plasma applications. *Approximate* MPC [13], which hinges on approximating MPC laws via offline computations of the optimal control problem, enables control of CAP devices at kHz sampling rates [157]. Deep neural network (DNN)-based approximations of MPC laws are especially attractive due to their low memory footprint and versatile embedded implementations on resource-limited, specialized hardware such as field programmable gate arrays (FPGAs) [72,158]. To this end, we present a multi-objective BO (MOBO) strategy for data-efficient and "globally" optimal adaptation of DNN-based control policies in a run-to-run manner. MOBO uses probabilistic surrogate models of multiple closed-loop performance measures (i.e., plasma treatment outcomes) to systematically trade off between exploration and exploitation of a subset of DNN parameters. The selection of this subset of parameters is guided by a global sensitivity analysis that quantifies the influence of each network parameter on the performance measures. As such, MOBO yields a data-efficient scheme for performance-oriented adaptation of DNN-based control policies. We experimentally demonstrate the proposed strategy for adaptive DNN-based approximate MPC of a CAP jet (CAPJ) with prototypical applications in processing of heat-sensitive biomaterials.

5.2.1 Robust MPC of Cold Atmospheric Plasma Jet

In this section, we present the control problem for a prototypical CAPJ in the context of personalized plasma treatments. This section revisits several aspects of the previous chapters. First, we use the CAPJ system described in Chapter 2.2, which is modeled by a linear time-invariant (LTI) state-space model

$$x_{k+1} = Ax_k + Bu_k, \tag{2.1a revisited}$$
$$y_k = Cx_k + Du_k, \tag{2.1b revisited}$$

and the thermal effect of the CAP is given by

$$\text{CEM}_{k+1} = \text{CEM}_k + K^{(T_{\text{ref}} - T_{s,k})}\delta t. \tag{2.3 revisited}$$

The goal of the plasma treatment is to deliver a desired amount of plasma effects as quickly as possible without violating comfort and safety constraints. Here, we revisit the robust MPC formulation, scenario-based MPC (sMPC) [146]. The optimal control problem is formulated as

$$\min_{u_{i,j|k}} \sum_{j=1}^{S} \omega_j \left[\sum_{i=0}^{N_p-1} V(x_{i,j|k}, u_{i,j|k}) + V_f(x_{N_p,j|k}) \right], \tag{4.19a revisited}$$
$$\text{s.t. } x_{i+1,j|k} = Ax_{i,j|k} + Bu_{i,j|k} + w_{i,j|k}, \tag{4.19b revisited}$$
$$(x_{i,j|k}, u_{i,j|k}) \in \mathcal{X} \times \mathcal{U}, \tag{4.19c revisited}$$
$$\textstyle\sum_{j=1}^{S} \tilde{E}_j U_{j|k} = 0, \tag{4.19d revisited}$$
$$x_{0,j|k} = x_k, \tag{4.19e revisited}$$
$$\forall i \in \{0, \dots, N_p - 1\},\ \forall j \in \{1, \dots, S\},$$

The solution to (4.19) defines the sMPC law as

$$\pi_{\text{smpc}}(x_k) = u^\star_{0|k}, \tag{5.3}$$

where $u^\star_{0|k}$ is the optimal first input. Here, the control objective is defined as the terminal cost

$$V_f(\text{CEM}_{N_p}) = (\text{CEM}_{sp} - \text{CEM}_{N_p})^2, \tag{5.4}$$

where CEM_{sp} denotes the setpoint CEM dose.

To adapt the policy for personalized CAP treatments, we focus on two closed-loop performance measures: (i) the delivery of a desired amount of thermal dose and (ii) the adherence to a comfort/safety constraint. We define (i) as a CEM tracking cost over the treatment time T

$$\phi_1 = \sum_{k=0}^{T} (\text{CEM}_{sp} - \text{CEM}_k)^2, \tag{5.5}$$

and (ii) as the sum of the degree of constraint violation in surface temperature over T

$$\phi_2 = \sum_{k=0}^{T} ([T_{s,k} - T_{\text{tol}}]^+)^2, \tag{5.6}$$

where T_{tol} is the nominal tolerated temperature constraint, and $[T_{s,k} - T_{\text{tol}}]^+$ is the positive magnitude of constraint violation. These measures are competing since T_{tol} is often set to a value at or near 43°C, which limits the rate of CEM delivery.

5.2.2 Approximate MPC using Deep Learning

The requirements of embedded control on low-cost, resource-limited hardware for point-of-use CAPJ applications pose a key challenge to online deployment of the sMPC law (5.3). The challenge arises from the high computational cost of the scenario-tree optimization in (4.19). To this end, we use DNNs to approximate (5.3).

Consider a dataset in the form of

$$\mathcal{T} = \{(x_q, \pi_{\text{smpc}}(x_q))\}_{q=1}^{n_s}, \tag{5.7}$$

representing n_s state-action (optimal input) pairs as acquired by the offline solution of (4.19). Let a DNN-based policy be defined as $\pi_{\text{dnn}} : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$. A generic feedforward description of a DNN constitutes a nonlinear, input-output mapping, where information is propagated from the input layer to the output layer via L hidden layers that contain H nodes [159]. As was given in Chapter 4.3.1, a DNN can be defined as

$$\pi_{\text{dnn}}(x;\theta) = \alpha_{L+1} \circ \beta_L \circ \alpha_L \circ \cdots \beta_1 \circ \alpha_1(x), \tag{4.2 revisited}$$

where θ is decomposed into two categories of parameters, those that relate to the weights θ_W and those that relate to the architecture θ_A. Specifically, $\theta_W = \{W_1, b_1, \dots, W_{L+1}, b_{L+1}\}$ denotes the collection of weights and biases that parameterize the network for a fixed architecture θ_A (e.g., type of activation function, L, and H).

The DNN parameters θ_W are generally fit by minimizing a mean-squared-error loss function. Meanwhile, the DNN hyperparameters θ_A related to its architecture (e.g., L, H, $\{\sigma_i\}_{i=1}^{L}$), as well as the training/fitting options (e.g., learning rate, optimizer solver) must be tuned. Tuning is a crucial step of the DNN policy training since the hyperparameters affect the resource utilization of hardware, e.g., the memory required to store the weights/parameters θ_W and the accuracy of the approximation of (5.3). While BO is commonly used to facilitate hyperparameter tuning, i.e., tuning θ_A, this work focuses on using BO to adapt DNN parameters θ_W. Here, the DNN is trained using closed-loop data as described in, e.g., [72]. This way, each step of the closed-loop trajectory is a solution to (5.3) and represents a suitable situation in which the closed-loop system is likely to operate.

Remark 4 *Adaptation of DNN-based control policies using data-driven optimization methods such as BO is prone to the curse of dimensionality. Thus, we utilize a global sensitivity analysis (GSA) [160] with respect to the performance measures ϕ_1 and ϕ_2 to decide which candidate parameters $\theta_{W,0} \subset \theta_W$ should be prioritized for control policy adaptation. The two main components of a GSA are the quantities of interest (QoI) to be analyzed and the corresponding inputs. A suitable set of QoIs are the closed-loop performance measures, $\hat{\phi}_1$ and $\hat{\phi}_2$, which serve as proxies to the true ones. By extensive sampling of the DNN parameters (i.e., inputs of GSA) we can evaluate the impact of DNN parameters on the performance measures (i.e., outputs of GSA) and, accordingly, determine which inputs are most responsible for the variability of the latter.*

5.2.3 Multi-objective Bayesian Optimization for Control Policy Adaptation

The control policy adaptation can be cast as a multi-objective (MO) problem characterized by M closed-loop performance measures $\{\phi_m\}_{m=1}^{M}$; see (5.5), (5.6). We denote the closed-loop system uncertainties by $\mathbf{d} = \{d(0), \dots, d(N)\}$. Further, we define a vector-valued performance measure as $\boldsymbol{h}(\theta) : \mathbb{R}^{n_\theta} \to \mathbb{R}^M$, with components $h_m(\theta) = \mathbb{E}_{\mathbf{d}}\left[\phi_m(\theta, \mathbf{d})\right]$, where $\theta \in \Theta$ are real-valued decision variables, namely the subset of DNN parameters that are adapted. Then, the MO optimization problem for optimal selection of θ is formulated as

$$\min_{\theta \in \Theta} \quad \boldsymbol{h}(\theta). \tag{5.8}$$

We approximate the expectation of each performance measure in a sample-based fashion as

$$h_m(\theta) = \mathbb{E}_{\mathbf{d}}\left[\phi_m(\theta, \mathbf{d})\right] \approx \frac{1}{N_\mathrm{d}} \sum_{j=1}^{N_\mathrm{d}} \phi_m(\theta, \mathbf{d}_j), \tag{5.9}$$

where N_d is the number of samples for a given θ. The sample-averaged approximation (5.9) yields noisy estimates of the performance measures

$$\psi_m(\theta) = h_m(\theta) + \epsilon^m, \tag{5.10}$$

where ϵ^m represents the noise of the m-th performance measure, and $\boldsymbol{\Psi}(\theta) = \{\psi_m(\theta)\}_{m=1}^{M}$ denotes the set of observed performance measures for a given θ.

Problem (5.8) cannot be directly solved in the case of expensive and black-box performance measures. Hence, the general idea of BO is to learn probabilistic surrogate models, typically Gaussian Processes (GPs), for the performance measures and select a set of points that jointly optimize the expected value of the current surrogates. This is done by solving a proxy problem where an acquisition function proposes points to query to refine the surrogate representing the performance measures. The querying strategy is based on the exploration/-exploitation trade-off: we look to query the measures at points that lie in a neighborhood that can contain the optima while also reducing the prediction uncertainty of the surrogate models. Given newly observed data $\mathcal{D} = \{(\theta_i, \boldsymbol{\Psi}_i)\}_{i=1}^{N_o}$, each performance measure is updated using Bayesian inference; e.g., for GP surrogates, GP regression is used [90].

Moreover, in a MO setting, there is not a single best optimizer since the performance measures can be conflicting. Hence, the goal is to discover a set of optimal points, a *Pareto frontier* comprised of *Pareto* optimal points. The Pareto frontier is a boundary in the performance measure space in which improving one measure is realized at the expense of degrading the others. Let us denote a set of Pareto optimal solutions as $\mathcal{P}$. Solutions contained within $\mathcal{P}$ are known to be *non-dominated* by other solutions in the feasible region. For control policy adaptation, *Pareto dominance* is defined in Definition 1.

Definition 1 *Given a set of parameters and their corresponding performance measures* $\{\theta, \boldsymbol{h}(\theta)\}$*, a solution* $\boldsymbol{h}(\theta_A)$ *dominates another solution* $\boldsymbol{h}(\theta_B)$ *if* $h_i(\theta_A) \leq h_i(\theta_B)$ $\forall i \in \{1, \dots, M\}$ *and* $\exists i \in \{1, \dots, M\}$ *such that* $h_i(\theta_A) < h_i(\theta_B)$*. Pareto dominance is denoted by* $\boldsymbol{h}(\theta_A) \prec \boldsymbol{h}(\theta_B)$*, while a solution* $\boldsymbol{h}(\theta_A)$ *is non-dominated if* $\nexists \theta_B \in \Theta$ *such that* $\boldsymbol{h}(\theta_B) \prec \boldsymbol{h}(\theta_A)$*.*

Given Definition 1 and the set of Pareto optimal performance measures (the Pareto frontier) is given as

$$\mathcal{P} = \{\boldsymbol{h}(\theta) \text{ s.t. } \nexists \theta^\star \in \Theta : \boldsymbol{h}(\theta^\star) \prec \boldsymbol{h}(\theta)\}, \tag{5.11}$$

and the Pareto optimal parameter set is given as

$$\vartheta = \{\theta \in \Theta \text{ s.t. } \boldsymbol{h}(\theta) \in \mathcal{P}\}. \tag{5.12}$$

Establishing a Pareto frontier will enable the selection of optimal control policies, each of which yields optimal performance with varying levels of trade-offs between performance measures.

In MOBO, the search for the Pareto frontier is commonly facilitated by the expected *hypervolume* improvement (HVI) acquisition function [135]. The expected HVI relies on the definition of an indicator that quantifies the Pareto optimality of the estimated Pareto frontier known as the hypervolume (HV).

Definition 2 *The HV is defined with respect to a reference point* $r \in \mathbb{R}^M$ *in the performance measure space. For a finite, estimated Pareto set* $\mathcal{P}$*, the HV is given as the* M*-dimensional*

Table 5.1: Sensitivity values (mean ± standard error) of the closed-loop measures to various parameters of the policy

	First Layer	**Last Layer**
Thermal Dose Delivery (ϕ_1)	$0.025 \pm 3.2 \times 10^{-4}$	$0.026 \pm 8.3 \times 10^{-4}$
Temperature Constraint (ϕ_2)	$0.036 \pm 4.7 \times 10^{-4}$	$0.038 \pm 1.1 \times 10^{-3}$

Lebesgue measure Λ_M of the space dominated by $\mathcal{P}$ and bounded by r

$$\mathcal{HV}(\mathcal{P}, r) = \Lambda_M \left(\bigcup_{i=1}^{|\mathcal{P}|} [r, \mathbf{\Psi}_i] \right), \tag{5.13}$$

where $|\mathcal{P}|$ is the cardinality of $\mathcal{P}$, and $[r, \mathbf{\Psi}_i]$ is the hyper-rectangle formed by the points r and $\mathbf{\Psi}_i$ [135].

The HV acts as a metric to quantify the quality of the Pareto frontier and is affected by the selection of the reference point. Thus, "convergence" to a single HV value means that MOBO has performed enough sampling (based on budget) to construct the best possible Pareto frontier, which is not necessarily the true one. Then, the HVI describes the incremental improvement of the HV of $\mathcal{P}$ if a new point is added. The HVI of a set of newly observed measures $\mathbf{\Psi}'$ is given by

$$\mathcal{HVI}(\mathbf{\Psi}', \mathcal{P}, r) = \mathcal{HV}(\mathbf{\Psi}' \cup \mathcal{P}, r) - \mathcal{HV}(\mathcal{P}, r). \tag{5.14}$$

Hence, the expected HVI acquisition function α_{EHVI} describes the expectation of HVI over the posterior of the performance measures and is given as

$$\alpha_{\text{EHVI}}(\theta) = \mathbb{E}[\mathcal{HVI}(\mathbf{\Psi}', \mathcal{P}, r)]. \tag{5.15}$$

Finally, to account for noisy observations of performance measurements, the noisy expected HVI acquisition (NEHVI) is employed; see [135]. Here, NEHVI is maximized with respect to the DNN parameters θ.

5.2.4 Adaptive DNN-based Control Policies for Personalized Plasma Treatments

We demonstrate the proposed MOBO strategy for control policy adaptation on the CAPJ described in Chapter 2.2.

Control Policy Approximation

First, we solved the sMPC problem (4.19) in closed loop to gather training data for approximating the initial control law (5.3). In (4.19), we set the prediction horizon $N_p = 5$,

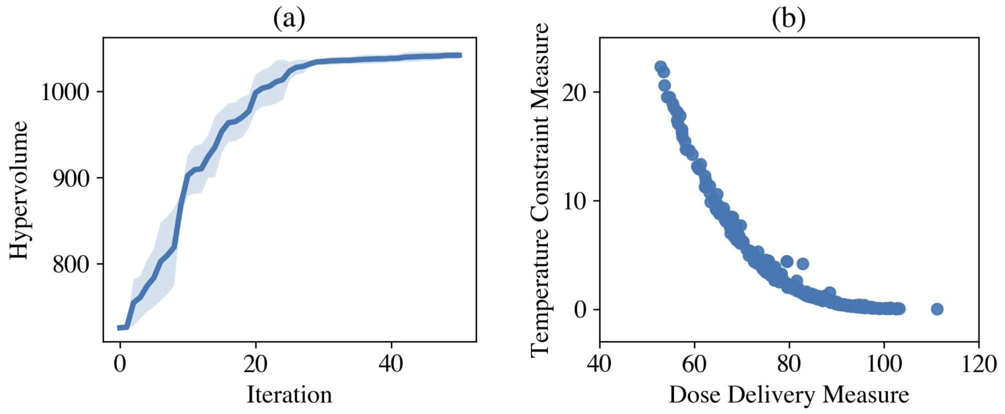


Figure 5.1: (a) Hypervolume improvement (mean ± two standard errors) and (b) observed Pareto frontier over five replicate runs of MOBO. The hypervolume improvement (a) demonstrates that MOBO reaches some optimal representation of the Pareto frontier. The Pareto frontier (b) demonstrates the trade-off between the competing performance measures (dose delivery: reducing treatment time; temperature constraint: satisfying patient comfort and safety).

the robust horizon $N_r = 2$, and the discrete uncertainty scenarios as $[0.01w_{\min}, 0, 0.01w_{\max}]$. The control inputs are constrained by $P \in [1.5, 5]$ W and $q \in [1.5, 5]$ SLM, and the states are constrained by $T \in [25, 45]$°C and $I \in [20, 80]$ arb. units. The sMPC was formulated using CasADi [139] and solved with IPOPT [102]. We simulated the true system with a mismatch between the plant and control model and normally distributed measurement noise $\mathcal{N}(0, (0.1)^2)$. We collected $n_s = 5,000$ samples of state-to-optimal-input mappings and trained a fully-connected feedforward DNN architecture with $L = 4$, $H = 7$, and ReLU activation functions. We trained the DNN for $5,000$ epochs using PyTorch [161] with the default optimizer settings. The resulting DNN-based policy achieved nearly equivalent performance to the implicit sMPC law. Furthermore, the computation time of the DNN, which depends on the architecture of the DNN, compared to solving (4.19), on a standard CPU (2.4 GHz quad-core Intel i5 processor) was roughly three orders of magnitude faster ($\sim 10^{-5}$ s versus $\sim 10^{-2}$ s).

Control Policy Adaptation in Closed-loop Simulations

We consider the treatment of a subject with characteristics that differ from the mean population values, and our goal is to adapt the initial policy designed for the population mean to cater to the individual subject. Note that the closed-loop performance measures are parameterized by subject-specific characteristics, namely K in the CEM setpoint tracking cost (5.5) and T_{tol} in the comfort constraint cost (5.6). Here, we examine the case in which the parameters of the population mean are $K_{\text{pop}} = 0.5$ and $T_{\text{tol,pop}} = 45$°C, while the characteristics of

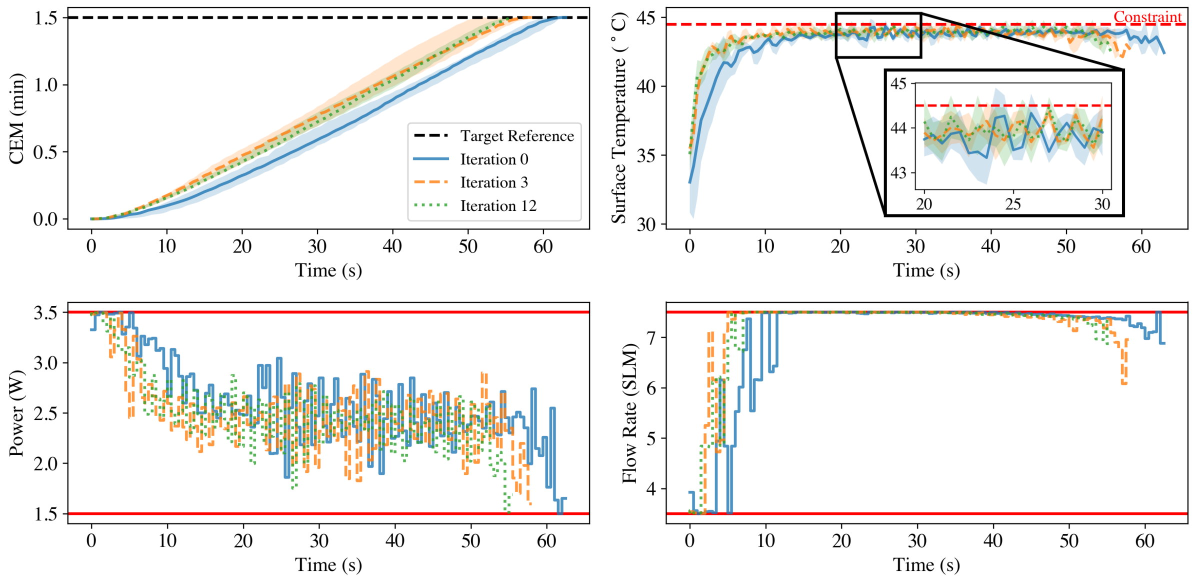


Figure 5.2: State and input profiles of closed-loop experiments at various iterations of MOBO. Each iteration of MOBO consisted of triplicate experiments. The CEM profile (upper left) shows the median value (solid line) along with the min/max range (shaded region). The surface temperature profile (upper right) shows the mean value (solid line) and two standard errors (shaded region). For the manipulated inputs (power and flow rate), only the mean value is plotted. The selected profiles shown are designated as the trajectories that correspond to the "incumbent best" policy parameterizations. The incumbent best is deemed as the initial policy, if a Pareto frontier cannot be established (i.e., in the first few iterations) or the policy parameterization on the Pareto frontier with the lowest temperature constraint measure.

the individual subject are $K_{\text{indiv}} = 0.55$ and $T_{\text{tol,indiv}} = 44.5°\text{C}$.

First, we examine the sensitivity of the DNN-based policy to perturbations in different subsets of its parameters. Knowing that the desire for personalized treatments is to minimize the number of trial-and-error treatments, we adapted a subset of DNN parameters due to its high dimensionality ($n_{\theta_W} = 212$). Common practice is to freeze the DNN and adapt the last layer and/or append a new layer to the network to adapt. To evaluate this practice, we examine the sensitivity of the closed-loop performance measures to the parameters of the first and last layers of the DNN-based policy. To perform a GSA as described in Remark 4, we used the sensitivity analysis tools by UQLab [162]. We used a moment-independent method (i.e., Borgonovo indices) to analyze the global sensitivity of the selected DNN parameters to the closed-loop performance measures. We generated $10,000$ samples of the 44 parameters encapsulated by the first and last layers of the 4-layer, 7-node DNN. Samples were selected from geometrically-bounded values from the initial policy parameters. For each sample, we ran triplicate closed-loop simulations using the DNN-based policy and evaluated the mean plus and minus standard error values of the observed closed-loop measures. Table 5.1 lists the results of the GSA. In general, the dose delivery measure (5.5) is less sensitive to changes in the parameters compared to the temperature constraint measure (5.6). Overall, both of the measures are equally sensitive to all of the parameters selected for GSA. Despite this, the parameters of the last layer have slightly higher influence with fewer number of parameters. Hence, we selected the last layer of the DNN-based policy as the subset of parameters to modify in our policy search procedure (i.e., $\theta = \boldsymbol{W}_{L+1}$ such that $n_\theta = 16$).

As a global optimization method, MOBO provides a means to systematically explore and detect trade-offs between competing performance measures. Fig. 5.1 shows the results of 5 replicates of 50 iterations of MOBO on a simulated CAPJ, where one iteration of MOBO is comprised of $N_{\mathbf{d}} = 3$ replicates.[2] The HV profile in Fig. 5.1(a) shows the "convergence" of MOBO. The reduced improvement in the HV after more than 30 iterations indicates that MOBO has achieved some optimal representation of the Pareto frontier depicted in Fig. 5.1(b). While it took 20 or more iterations to achieve this Pareto frontier, the first few iterations of MOBO can drastically improve upon the initial policy. The steep increase in HV suggests that the initial policy parameterization is suboptimal, and a new Pareto optimal point can be found in the first few iterations even when starting with a suboptimal solution.

Control Policy Improvement for Real-time Treatments

For the experimental demonstrations of the proposed approach on the CAPJ depicted in Fig. 2.1, the sMPC had $N_p = N_r = 2$, and the input bounds were adjusted to $P \in [1.5, 3.5]$ W and $q \in [3.5, 7.5]$ SLM. Then, 11 closed-loop experiments resulting in $n_s = 1,378$ samples were performed to gather training data for the DNN approximation. The DNN was trained with the same structure and procedure as described for the simulation studies and achieved similar closed-loop performance to implicit sMPC. MOBO was performed for 15 total iterations due to a limited budget of 45 treatments.

Fig. 5.2 shows the state and input profiles of 3 particular iterations of MOBO. Within each iteration, we performed $N_{\mathbf{d}} = 3$ replicate real-time experiments to account for the intrinsic variability of the system. The CEM profiles are plotted with min-max bounds of the three replicates represented by the shaded region, while the solid line represents the median value of the triplicate runs. The temperature profiles represent the mean value (solid lines) plus and minus two standard errors (shaded region) of the triplicate experiments. Both input profiles are plotted with the mean value from the triplicate experiments. In Fig. 5.2, the profiles shown are determined to be a few of the "best" treatment options encountered

[2]To implement MOBO, we used Ax [123]. Ax interfaces with BoTorch [163] to perform BO, and BoTorch interfaces with GPyTorch [164] for the surrogate modeling with GPs. These tools were primarily used with their default settings, using the Matern 5/2 kernel for GPs and the noisy EHVI acquisition function. Codes are available at `https://github.com/kchan45/BO4Policy_Search_Plasma`.

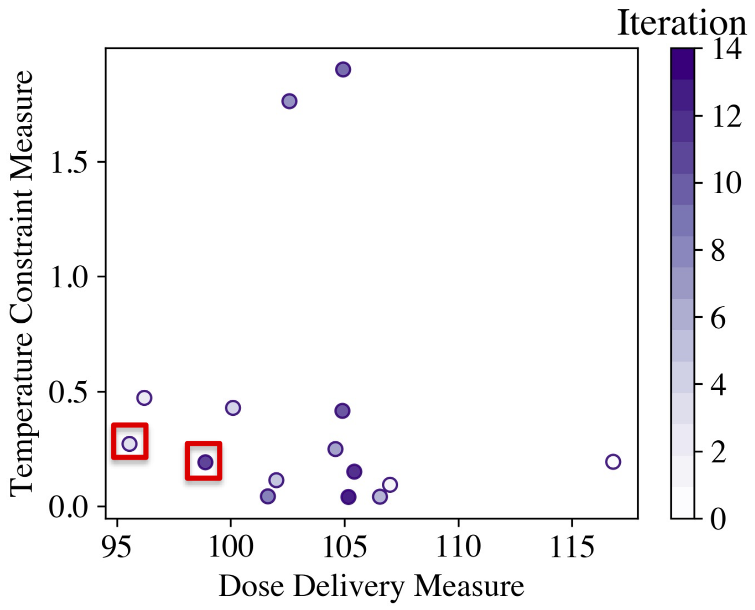


Figure 5.3: Observed performance measures from the MOBO exploration. A total of 15 iterations of MOBO were performed. Each individual point represents the mean performance measure values from triplicate real-time experiments at each iteration. The red boxes identify the estimated Pareto optimal points.

through the process of MOBO. In this case, the "best" is described in one of two ways: (i) if there was insufficient data to establish a clear Pareto frontier, then the best treatment was considered the initial policy, and (ii) once an estimated Pareto optimal point was found, the best treatment used the policy that produces the lowest constraint violation. This sequence of treatment protocols follows a "safe" treatment intuition. As in for (i), the initial treatment is deemed safe for the general population and is considered "best" for the time being. In the case of (ii), once a Pareto frontier is established, the treatment may then be switched to a more optimal one at the cost of minor temperature violations. Note that establishing some trade-off between the different performance measures via an estimated Pareto frontier allows for the personalization of plasma treatments.

From Fig. 5.2, the first "best" profile is the initial profile (in blue); a new "best" is encountered after Iteration 3 (in dashed orange). The dashed orange profile represents a new parameterization of the policy that outperforms the initial blue policy, as it achieves the CEM faster (reducing the median treatment time by roughly 8 s or 13%), with slight constraint violations. After more iterations of MOBO, a new policy in dotted green is found at Iteration 12. In this case, the CEM delivery on average is similar to the orange policy (reducing the median treatment time from the initial policy by 10 s or 16%), while maintaining a lower constraint cost. Furthermore, in Fig. 5.2, the flow rate of He tends to become saturated during treatment. In general, higher He flow rates are characteristic of lower temperatures. Thus, because of the temperature tolerance specification, the operation of the CAPJ necessitates higher flow rate to remain within the region of desirable operating temperatures. The locations of these "best" points in the performance measure space is shown by the red boxes in Fig. 5.3. With few iterations, the Pareto frontier cannot be visually established, but several points may still be identified as Pareto optimal by their proximity to the minimization of both performance measures. Fig. 5.3 is consistent with the profiles in Fig. 5.2 in that Pareto optimal points are found at Iterations 3 and 12 (as indicated by the red boxes). As such, a strategy has been established that can trade off between multiple performance measures in order to tailor the treatment to individual subjects.

5.3 Safe Exploration in Control Policy Tuning

This section introduces a novel method in safe data-driven optimization. Penalty-based safe BO methods [165] force the query points to remain in the interior of a partially-revealed safety region, which may result in unacceptable (and unquantified) performance losses. Our method aims to enhance the explorative capabilities of safe BO by re-introducing a metric for the enlargement of the feasible set. Note that this section is broadly applicable as a method development, so the notation used in this section is distinct from the rest of this dissertation.

5.3.1 Introduction to Safe Data-driven Optimization

Advances in data-driven control and decision-making capabilities have created significant opportunities for autonomy for vehicles, robots, and biomedical devices [166]. Interactions with humans make safety a fundamental requirement for these autonomous systems. Thus, the underlying problem for autonomy can be generally posed as a constrained optimization problem of the form

$$\min_{x \in \mathcal{X}} \left\{ f^0(x) : f^i(x) \geq 0, \ \forall i = 1, \ldots, m \right\}, \tag{5.16}$$

where x are the decision variables (i.e., modifiable parameters), $f^0 : \mathcal{X} \to \mathbb{R}$ is the objective function, $f^i : \mathcal{X} \to \mathbb{R}$ are constraints, and $\mathcal{X} \subset \mathbb{R}^{n_x}$ is some compact domain.

In autonomous systems, it is often the case that the mathematical structure of the objective $f^0(x)$ and constraints $f^i(x)$ are not exactly known, e.g., those derived from closed-loop trajectories. In such cases, we often refer to the functions $\{f^i(x)\}_{i=0}^m$ as "black-box" in the sense that they can only be learned from noisy observations at specific query points $x \in \mathcal{X}$. These observations must then be used in a strategy to compute an optimum for x. One such strategy is Bayesian optimization (BO) [70]. BO is a sequential decision-making strategy that uses probabilistic surrogate models of $\{f^i(x)\}_{i=0}^m$ to optimize a proxy problem to (5.16). The surrogate models, often represented with Gaussian Processes (GPs), are updated via

Bayesian inference [90]. Optimizing the proxy problem is facilitated by an acquisition function $\alpha(x) : \mathcal{X} \to \mathbb{R}$, which, in some form, leverages the uncertainty in the posterior model. This feature of optimizing with respect to both the belief of the optimum and the uncertainty surrounding this belief is commonly referred to as the trade-off between exploration and exploitation. By iteratively querying and updating the surrogates, BO systematically explores the design space $\mathcal{X}$ to find an optimum.

For the safe interaction of autonomous systems with humans, it is imperative to ensure that the proposed parameter choices satisfy constraints $\{f^i(x)\}_{i=1}^m$. In other words, evaluating any arbitrary x in $\mathcal{X}$ can lead to constraint-violating designs that can have dangerous outcomes. Yet, identifying a safe, or feasible, design space $\mathcal{F} \subseteq \mathcal{X}$ when the constraints are unknown is challenging. To this end, several recent works have been proposed in the realm of safe BO. At its core, the safe BO problem is exactly the constrained optimization problem (5.16) with strict adherence to the constraints. Two common approaches are taken: (i) a penalty-based strategy, where the constraints act as a penalty term in the objective; and (ii) a safe set strategy, where points are only chosen from an estimation of the safe region. In [165], the acquisition function is augmented with barrier functions, a take on (i); it uses the posterior estimates of the constraints to directly penalize the acquisition objective to limit the search to revealed safe points. Here, the solution will only be locally optimal near the initial safe design point x_0. Alternatively, [167] uses the posterior models to compute a partially-revealed safe set using Lipschitz continuity properties. This safe set is further subdivided into a set of potential "optimizers" and a set of potential "expanders." Then, the most uncertain element from the combination of the optimizers and expanders is suggested as the next query. This method takes an exploration-first perspective to maximize the discovery of safe points and then switches focus to the standard exploitation/exploration trade-off. This method may lead to "wasteful" queries within the feasible region.

In an attempt to expand the feasible region, [168] proposes an ϵ-greedy approach to switch to a pure exploratory strategy to directly explore the boundary of the feasible region. However, this exploratory procedure remains agnostic to improvements in the main objective. This section presents SEBO (Safe Explorative Bayesian Optimization), a new safe BO method that avoids potential performance losses by incorporating information gained by expanding the feasible region. As depicted in Fig. 5.4, penalty-based strategies for safe BO may be prone to being overly conservative such that they may get stuck in the locally feasible region near the initial safe point. SEBO uses a relaxed formulation to widen the search space that more likely encapsulates the true optimum. Safety is ensured by projecting back to the estimated safe region, but at the same time, maximizing the potential to increase knowledge around the safe set in the direction of improvement. Thus, SEBO effectively incorporates directed information to explore the safe region(s).

We demonstrate SEBO for an example application for personalized plasma medicine, which is an emerging field involving the use of cold atmospheric plasmas (CAPs) for a variety of medical treatments [150]. Automated CAP treatments using advanced control (e.g., model predictive control, MPC) are necessary for ensuring effective delivery of plasma effects [19, 61]. Tailoring the plasma effects is key to ensuring the efficacy of plasma treatments [25].

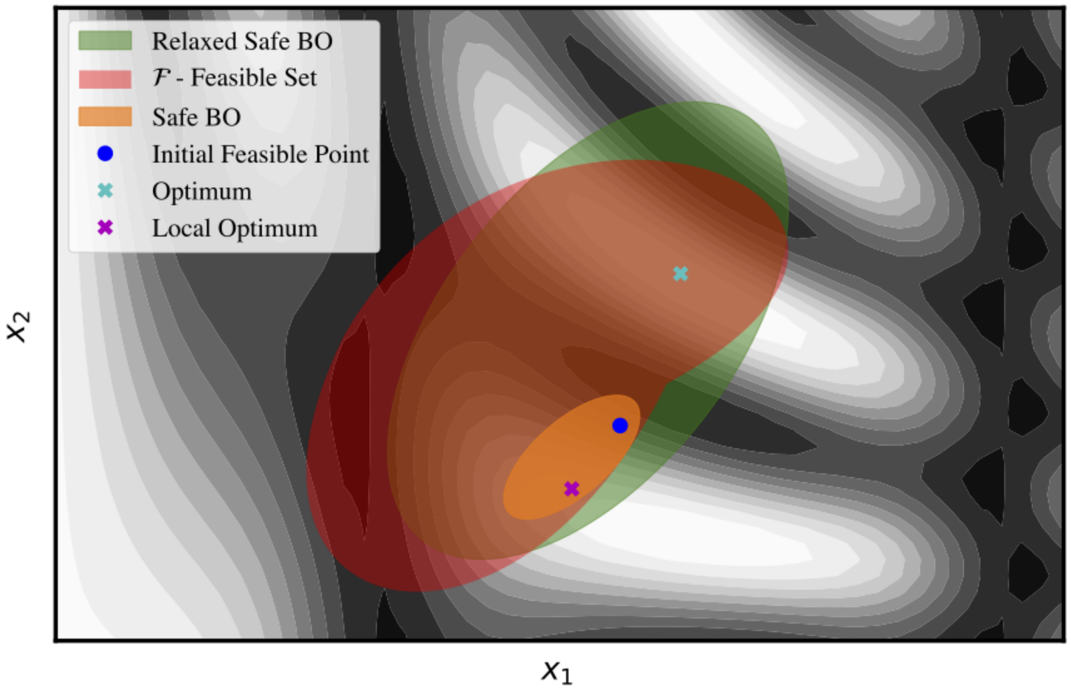


Figure 5.4: Exemplary feasible set using safe Bayesian optimization (BO) (in orange) versus a relaxed problem (in green). The true feasible set is in red, while the initial feasible point is in blue. The true optimum is denoted with a cyan "x", and a local optimum is denoted with a magenta "x". Contours of the objective are in gray-scale with more optimal spaces in white.

However, the underlying mechanisms of plasma-surface interactions can only be quantified for a population of subjects [19,169]. Therefore, iterative improvements in delivery of plasma effects using BO will enable personalization of CAP treatments, wherein ensuring patient safety is of the utmost importance. We compare SEBO's performance to alternative safe BO approaches and demonstrate that it can mitigate getting stuck in a local feasible region while realizing safe CAP treatments.

5.3.2 Safe Bayesian Optimization using Logarithmic Barrier Functions

Since the model and constraint functions in (5.16) are unknown, we must learn them from data. Here, we focus on the case that these functions can be modeled as independent

Gaussian processes (GPs)

$$f^i(x) \sim \mathcal{GP}(\mu^i(x), k^i(x,x')), \ \forall i = 0, \dots, m, \tag{5.17}$$

where $\mu^i(x) = \mathbb{E}\{f^i(x)\}$ is the prior mean function and $k^i(x,x') = \mathbb{E}\{(f^i(x) - \mu^i(x))(f^i(x') - \mu^i(x'))\}$ is the prior covariance (kernel) function of the objective and safety constraints. GP models are non-parametric and have the property that the posterior model, conditioned on noisy observations $\boldsymbol{y}_n^i = (y_1^i, \dots, y_n^i)$ at query points $(x_1, \dots, x_n)$, remains a GP with the following analytic expressions for the mean, kernel, and standard deviation functions

$$\begin{aligned}
\mu_n^i(x) &= \boldsymbol{k}_n^{i\top}(x)(\boldsymbol{K}_n^i + \eta^i I)^{-1}\boldsymbol{y}_n^i, && (5.18a)\\
k_n^i(x,x') &= k^i(x,x') - \boldsymbol{k}_n^{i\top}(x)(\boldsymbol{K}_n^i + \eta^i I)^{-1}\boldsymbol{k}_n^i(x'),\\
\sigma_n^i(x) &= \sqrt{k_n^i(x,x)}, && (5.18b)
\end{aligned}$$

where $\boldsymbol{k}_n^i(x) = [k^i(x_1,x), \dots, k^i(x_n,x)]^\top$, $\boldsymbol{K}_n^i$ is the positive definite kernel matrix whose elements are given by $[\boldsymbol{K}_n^i]_{\nu,\omega} = k_i(x_\nu, x_\omega)$ for all $\nu, \omega \in \{1, \dots, n\}$, and $\eta^i > 0$ is the variance of a zero-mean Gaussian noise model for the observations, i.e., $y_j^i = f^i(x_j) + \epsilon_j^i$ for some ϵ_j^i that is R-sub Gaussian noise [170].

If the GP models are sufficiently "well-calibrated," then they can provide high probability confidence bounds on $\{f^i(x)\}_{i=0}^m$. We summarize this requirement below.

Assumption 1 (Well-calibrated GPs) *The GP models for the unknown objective and constraint functions* $\{f^i(x)\}_{i=0}^m$ *satisfy the inequality below* $\forall x \in \mathcal{X}$, $n \geq 0$, *and* $i = 0, \dots, m$

$$|f^i(x) - \mu_n^i(x)| \leq \sqrt{\beta_{n+1}^i}\sigma_n^i(x), \tag{5.19}$$

with probability at least $1 - \delta$ *for some* $\delta \in (0,1)$.

This assumption can be satisfied by properly selecting the sequence of confidence bound parameters $\{\beta_{n+1}^i\}_{n\geq 0}$ as long as the functions $\{f^i(x)\}_{i=0}^m$ have a bounded reproducing kernel Hilbert space (RKHS) norm; see [170] for expressions for $\{\beta_{n+1}^i\}_{n\geq 0}$, which are defined in terms of the maximum information gain.

For convenience, we rewrite Assumption 1 in terms of the lower confidence bound (LCB) and upper confidence bound (UCB) on the unknown functions at iteration n

$$l_n^i(x) = \mu_n^i(x) - \sqrt{\beta_{n+1}^i}\sigma_n^i(x), \tag{5.20a}$$

$$u_n^i(x) = \mu_n^i(x) + \sqrt{\beta_{n+1}^i}\sigma_n^i(x), \tag{5.20b}$$

where the inequality (5.19) can be equivalently stated as $f^i(x) \in [l_n^i(x), u_n^i(x)]$ using the LCB/UCB definitions. The main idea behind a safe BO procedure is then to sequentially select new sample points $x_1, x_2, \dots$ that have a high probability of satisfying safety constraints at every iteration. This differs from standard BO that would query by solving $x_{n+1} \in \arg\min_{x\in\mathcal{X}} l_n^0(x)$ (using an LCB acquisition function). As proposed in [165], one can ensure x_{n+1} remains in the interior of the partially-revealed safe set by solving

$$x_{n+1} \in \arg\min_{x\in\mathcal{X}} \left\{ l_n^0(x) + \tau \textstyle\sum_{i=1}^{m} \mathcal{B}_{l_n^i}(x) \right\}, \tag{5.21}$$

where $\mathcal{B}_g(x) = -\ln(g(x))$ is the logarithmic barrier applied to a constraint $g(x) \geq 0$ and $\tau > 0$ is a tunable parameter that ensures the barrier term converges to the exact indicator penalty function in the limit $\tau \to 0$. Notice that (5.21) accounts for both performance and safety. The safety guarantees conferred by (5.21) are summarized in the following theorem.

Theorem 1 (Safe Learning [165]) *Let Assumption 1 hold, the feasible set $\mathcal{F} = \{x \in \mathcal{X} : f^i(x) \geq 0, \forall i = 1, \dots, m\}$ be non-empty, and there exists at least one known safe starting point $x_0 \in \mathcal{F}$. Then, the sequence of query points $\{x_n\}_{n\geq 1}$ generated by* (5.21) *satisfies*

$$Pr\left\{ f^i(x_n) \geq 0,\ \forall i = 1, \dots, m, \forall n \geq 1 \right\} \geq 1 - \delta, \tag{5.22}$$

for any chosen $\delta \in (0, 1)$.

The proof of this theorem is based on three key arguments. First, the partially-revealed feasible region, defined by

$$\hat{\mathcal{F}}_n = \{x \in \mathcal{X} : l_n^i(x) \geq 0,\ \forall i = 1, \dots, m\}, \tag{5.23}$$

must be contained within the true feasible region $\hat{\mathcal{F}}_n \subseteq \mathcal{F}$ with high probability. Second, $\hat{\mathcal{F}}_n \neq \emptyset$ must be non-empty given a known safe point x_0. Third, the log-barrier term in (5.21) always guarantees the next query point x_{n+1} is contained in this estimated region, i.e., $x_{n+1} \in \hat{\mathcal{F}}_n$.

The primary challenge encountered by this type of safe BO approach is that it does not have any direct incentive to grow the size of the partially-revealed safety region $\hat{\mathcal{F}}_n$. Thus, it may become "stuck" in the sense that $\hat{\mathcal{F}}_n \subset \mathcal{F}$ as $n \to \infty$, which could lead to sub-optimal performance in cases where the global solution to (5.16) satisfies $x^\star \in \mathcal{F} \setminus \hat{\mathcal{F}}_n$. We look to overcome this challenge by introducing SEBO.

5.3.3 Safe Explorative Bayesian Optimization

The main idea motivating the proposed SEBO method is that there is value in enlarging the certifiable safety region at every iteration of BO to ensure that we do not get stuck in a sub-optimal solution. Thus, we require a metric that, when evaluated at any $x \in \mathcal{X}$, provides a reasonable measure of the potential benefit of querying the constraints at that point in the future. Let $\text{Vol}(\hat{\mathcal{F}}_n)$ denote the volume of the partially-revealed safe set. We

Algorithm 1 Safe Explorative Bayesian Optimization

Require: Domain $\mathcal{X}$, safe point $x_0 \in \mathcal{F}$, initial data $\mathcal{D}_0 = \{x_0, \{f^i(x_0)\}_{i=0}^m\}$, $m+1$ GP models (5.17), confidence bound parameters $\{\beta_{n+1}^i\}_{n\geq 0}$, barrier parameter $\tau > 0$, switching tolerance $\varepsilon \geq 0$, and exploration radius $b \geq 0$.

```
1:  for $n = 0, 1, \dots$ do
2:      $x_{n+1} \leftarrow \arg\min_{x\in\mathcal{X}} \{l_n^0(x) + \tau \sum_{i=1}^m \mathcal{B}_{l_n^i}(x)\}$
3:      if $\sigma_n^0(x_{n+1}) \leq \varepsilon$ then
4:          $x_{n+1}^r \leftarrow \arg\min_{x\in\mathcal{X}} \{l_n^0(x) + \tau \sum_{i=1}^m \mathcal{B}_{u_n^i}(x)\}$
5:          $x_{n+1}^p \leftarrow \arg\min_{x\in\hat{\mathcal{F}}_n} \|x - x_{n+1}^r\|$
6:          $x_{n+1} \leftarrow \arg\max_{x\in\partial\hat{\mathcal{F}}_n \cap N_b(x_{n+1}^p)} \sum_{i=1}^m \sigma_n^i(x)$
7:      end if
8:      Query $x_{n+1}$ and observe objective and constraints
9:      Update data $\mathcal{D}_{n+1} \leftarrow \mathcal{D}_n \cup \{x_{n+1}, \{f^i(x_{n+1})\}_{i=0}^m\}$
10:     Update GP models with $\mathcal{D}_{n+1}$ using (5.18)
11: end for
```

can propose the following safety-based acquisition function that we refer to as the expected safety improvement (ESI)

$$\text{ESI}_n(x) = \mathbb{E}_n\left\{\text{Vol}(\hat{\mathcal{F}}_{n+1}) - \text{Vol}(\hat{\mathcal{F}}_n) \mid x_{n+1} = x\right\}, \tag{5.24}$$

where $\mathbb{E}_n\{\cdot\}$ denotes the expectation with respect to the posterior distribution given all function evaluations up until iteration n. There are two important challenges with (5.24): (i) it is expensive to compute and optimize since it requires repeated estimation of the volume of a set, though this can in principle be done with Monte Carlo methods (see, e.g., [171]); and (ii) any growth in the safety region is valued by this metric, which can impede the ability to discover new safe points that are likely to improve performance over multiple steps in the future. SEBO attempts to overcome both of these challenges by applying a series of steps that do not compromise the safety guarantees established in Theorem 1.

The first major step of SEBO is to decide when the choice in (5.21) is likely to contain low information content. The most straightforward approach is to check if $\sigma_n^0(x_{n+1}) \leq \varepsilon$, where $\varepsilon \geq 0$ is a user-specified tolerance value. This implies we can confidently predict the value of $f^0(x_{n+1})$ and, thus, have no additional room for improvement within $\hat{\mathcal{F}}_n$. Whenever such a situation occurs, we must explore outside the current safe region. To decide a new search direction, we solve the following relaxed problem

$$x_{n+1}^r \in \arg\min_{x\in\mathcal{X}} \left\{l_n^0(x) + \tau \textstyle\sum_{i=1}^m \mathcal{B}_{u_n^i}(x)\right\}, \tag{5.25}$$

where the LCB in the log-barrier term in (5.21) is replaced with the UCB. This change fundamentally alters the way that constraints are handled in the search process. In particular, (5.25) operates over a relaxed feasible region

$$\tilde{\mathcal{F}}_n = \{x \in \mathcal{X} : u_n^i(x) \geq 0, \ \forall i = 1, \dots, m\}. \tag{5.26}$$

Since $\tilde{\mathcal{F}}_n$ contains the global solution with high probability under Assumption 1 [172], sampling at $\{x_n^r\}_{n\geq 1}$ will guarantee convergence to the global solution; however, it will result in loss of the safety guarantees. Instead of directly sampling at this point, SEBO finds the closest safe point to x_{n+1}^r by solving the following projection problem

$$x_{n+1}^p \in \underset{x\in\hat{\mathcal{F}}_n}{\arg\min} \|x - x_{n+1}^r\|. \tag{5.27}$$

For any $x_{n+1}^r \notin \hat{\mathcal{F}}_n$, the projected point $x_{n+1}^p \in \partial\hat{\mathcal{F}}_n$ will lie on the boundary of the partially-revealed safe region and, thus, is more likely to expand the boundary. However, this projection does not account for the uncertainty of the constraint functions and may sample low uncertainty points. Thus, the final step of SEBO is to find the point with the largest sum of standard deviations in a neighborhood of the safe region boundary around x_{n+1}^p, i.e.,

$$x_{n+1}^s \in \underset{x\in\partial\hat{\mathcal{F}}_n\cap N_b(x_{n+1}^p)}{\arg\max} \textstyle\sum_{i=1}^m \sigma_n^i(x), \tag{5.28}$$

where $N_b(z) = \{x : \|x - z\| \leq b\}$ is a b-radius ball around point z. When the solution to (5.21) has low information, x_{n+1}^s is proposed as the new query point. The SEBO method is summarized in Algorithm 1. Although the practical performance of SEBO will be affected by the choice of parameters ε and b, they will not affect the safety properties, as summarized below.

Theorem 2 *Let the assumptions in Theorem 1 hold. Then, for any choice of* $\varepsilon, b \geq 0$, *the sequence of query points* $\{x_n^s\}_{n\geq 1}$ *generated by* (5.28) *will satisfy the safety constraints* (5.22)*, with* x_n *replaced by* x_n^s*, for any* $\delta \in (0,1)$.

Proof: The projection (5.27) and exploration (5.28) steps of SEBO ensure $x_{n+1}^s \in \hat{\mathcal{F}}_n$ for all $n \geq 0$ such that the same arguments used in the proof of Theorem 1 follow. ■

It is interesting to note that Algorithm 1 reduces to the safe BO method in [165] in the case that $\sigma_n^0(x_{n+1}) > \varepsilon$ always holds, which is guaranteed to be true whenever $\varepsilon = 0$. As such, we can interpret SEBO as a generalization of this method. SEBO will be particularly useful whenever one starts with a very restrictive inner approximation of $\mathcal{F}$.

5.3.4 Personalized Plasma Treatment Guidance

CAP Jet Modeling and Control

We consider a kHz-excited CAP jet (CAPJ) in helium [61] as described in Section 2.2. The manipulated inputs are applied power P (in Watts) and helium flow rate q (in standard liters per minute, SLM). The measured outputs are maximum surface temperature T (°C) and total optical intensity I (in arbitrary units) of the plasma at the plasma-surface incident point. The system dynamics $h(\cdot)$ are described by an observable, canonical form of a linear time-invariant model

$$s(k+1) = As(k) + Ba(k) + w(k), \tag{5.29}$$

where k is the discrete-time step, $s = [T, I]^\top \in \mathbb{R}^2$ is the vector of states, $a = [P, q]^\top \in \mathbb{R}^2$ is the vector of manipulated inputs, w is a stochastic variable that represents the overall system uncertainty, and A, B are the state-space matrices identified using subspace identification [173].

CAP treatments rely on the quantification of the delivered plasma effects to a surface. We describe the accumulation of thermal effects on a target with a metric called cumulative equivalent minutes (CEM) [1, 87] given by

$$\mathrm{CEM}(k+1) = \mathrm{CEM}(k) + K^{(T_{\mathrm{ref}} - T(k))}\delta t, \tag{5.30}$$

where $K > 0$ is an exponential base dependent on physical properties of the substrate, $T_{\mathrm{ref}} = 43°\mathrm{C}$ is the reference temperature, δt is the sampling time, and $\mathrm{CEM}(0) = 0$. Here, $\delta t = 0.5$ s in accordance with our open-loop data collection.

For a controlled plasma treatment, we use MPC, which is formulated in terms of the optimal control problem

$$\begin{aligned}
\min_{\mathbf{s}(k), \mathbf{a}(k)} \quad & (\mathrm{CEM}_{sp} - \mathrm{CEM}_c(N_p|k))^2 && (5.31\text{a})\\
\text{s.t.} \quad & s(i+1|k) = h_c\left(s(i|k), a(i|k)\right), && (5.31\text{b})\\
& (s(i|k), a(i|k)) \in \mathcal{S} \times \mathcal{A}, && (5.31\text{c})\\
& s(0|k) = s(k), && (5.31\text{d})\\
& \forall i \in \{0, \ldots, N_p - 1\},
\end{aligned}$$

where $\mathbf{s}(k) = [s(0|k)^\top, \ldots, s(N_p|k))]^\top$ is the vector of predicted states $s(i|k)$ over the prediction horizon $N_p = 5$ at time k; $\mathbf{a}(k) = [a(0|k), \ldots, a(N_p - 1|k))]$ is the vector of predicted inputs $a(i|k)$ at time k; CEM_{sp} is the setpoint for the CEM; $\mathcal{S} = [25°\mathrm{C}, 0 \text{ arb. units}] \times [45°\mathrm{C}, 80 \text{ arb. units}]$ is the set of state constraints; $\mathcal{A} = [1.5 \text{ W}, 1.5 \text{ SLM}] \times [5 \text{ W}, 5 \text{ SLM}]$ is the set of input constraints; $h_c(s, a) = \hat{A}s + \hat{B}a$ is the control-relevant state space model that may differ from (5.29); and $\mathrm{CEM}_c(N_p|k) = \mathrm{CEM}(k) + \sum_{i=0}^{N_p-1} \hat{K}^{T_{\mathrm{ref}} - T(i|k)}\delta t$ is the control-relevant CEM model that may differ from (5.30). The solution to (5.31) defines the MPC law as

$$\pi_c\left(s(k)\right) = a^\star(0|k), \tag{5.32}$$

where $a^\star(0|k)$ is the optimal first input. The MPC problem (5.31) is implemented in Python using CasADi [139] and is solved using IPOPT [102].

MPC Law Adaptation using SEBO

The plant (5.29) and (5.30) has parameters (A, B, K) that are specific to a given patient, which are not known in advance. The control method in (5.31), on the other hand, only has estimates of these values based on general population data. Therefore, to personalize a CAPJ

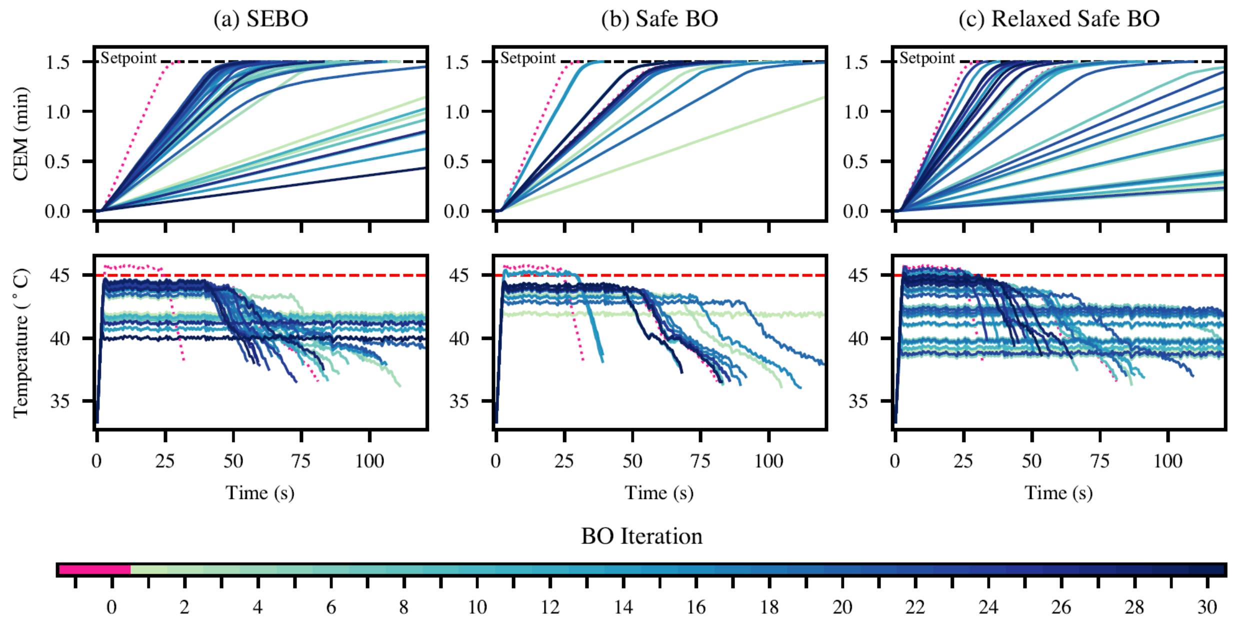


Figure 5.5: Comparison of observed closed-loop profiles between three strategies: (a) SEBO, (b) safe BO, and (c) the relaxed formulation of safe BO. The top figures represent the evolution of CEM over a treatment period of 120 s. The bottom figures represent the evolution of temperature over the same treatment period. The colors/gradient of the profiles indicate the evolution of the profiles over 30 iterations of BO. The first two profiles in dotted pink indicate the initial data provided to BO.

treatment, we propose to use SEBO to adapt a subset of these parameters to improve closed-loop performance while remaining safe using as few iterations as possible. This problem can be formulated in terms of (5.16) by defining the following black-box functions

$$f^0(x) = \textstyle\sum_{k=0}^{N}(\mathrm{CEM}_{sp} - \mathrm{CEM}(k))^2, \tag{5.33a}$$

$$f^1(x) = \textstyle\sum_{k=0}^{N}([T(k) - T_{\max}]^+)^2, \tag{5.33b}$$

where $x = [\hat{A}_{11}, \hat{A}_{12}, \hat{A}_{21}, \hat{A}_{22}, \hat{K}]^\top$ are the subset of model parameters in the controller that we allow to be modified (mainly the elements of the estimated A matrix and K constant since we assume B can be accurately identified), N denotes the treatment time, and $T_{\max}$ denotes the maximum allowed safe surface temperature. The objective (5.33a) corresponds to the cumulative deviation of the CEM value from its setpoint while the constraint (5.33b) corresponds to the squared summation of all constraint violations over the full course of the treatment. The CEM and temperature values in (5.33a) and (5.33b) correspond to the true closed-loop values obtained by applying the MPC law (5.32) to the plant (5.29)–(5.30). As such, every evaluation of these functions requires an expensive closed-loop experiment. We allow all model parameters to vary within the following geometric bounds,

$$\underline{x} = \begin{cases} \hat{x}_0/v, & \hat{x}_0 > 0, \\ v\hat{x}_0, & \hat{x}_0 \leq 0, \end{cases}, \quad \overline{x} = \begin{cases} v\hat{x}_0, & \hat{x}_0 > 0, \\ \hat{x}_0/v, & \hat{x}_0 \leq 0, \end{cases},$$

where $\hat{x}_0$ is the nominal value of x, $\underline{x}$ is the lower bound, $\overline{x}$ is the upper bound, and $v = 2$. These bounds define the search space $\mathcal{X}$.

Results

To demonstrate SEBO, we consider the CAP treatment of a subject over a time period of $N = 120$ s. The objective is to deliver $\text{CEM}_{sp} = 1.5$ min of thermal dose as quickly as possible, while the constraint with $T_{\max} = 45°\text{C}$ ensures safety and comfort of the individual subject. We consider a limited budget of 30 iterations for SEBO, where the objective (5.33a) and constraint (5.33b) are observed after each full treatment time of 120 s and are modeled as independent GPs.[3] We select $\tau = 10^3$ and $\{\tilde{\beta}^i\}_{i=0}^m = 0.1$ as "poor" selection of the parameters from the standard safe BO, while $\varepsilon = 0.5$ and $b = 10^{-2}$ in Algorithm 1.[4] ε is chosen based on a user preference to indicate the level of desired potential improvement. b is chosen to be a small number near the neighborhood of projected query x_{n+1}^p. In this work, the sets $\hat{\mathcal{F}}_n$ and $\partial\hat{\mathcal{F}}_n$ are approximated using random samples.

Observed closed-loop trajectories of the CAP treatment for three BO methods are shown in Fig. 5.5. The three methods are compared column-wise: (a) SEBO, (b) standard safe BO (Section 5.3.2), and (c) relaxed safe BO (BO using only (5.25)). The initial dataset $\mathcal{D}_0$ consists of one known feasible and one known unfeasible point in $\mathcal{X}$, shown in dotted pink. An infeasible initial data point helps to initially delineate between a safe region and unsafe region. In practice, such data points are readily estimated. For example, a controller that operates with low power at all times will most certainly provide a feasible solution, as power is directly related to surface temperature. Meanwhile, a controller that operates with high power at all times will most certainly provide an infeasible solution. The remaining profiles evolve over 30 iterations. Yellow-green profiles indicate earlier observations, and dark blue profiles indicate later observations. The top figures are the CEM profiles, which represent the objective (5.33a), and the bottom figures are the temperature profiles, which represent the constraint (5.33b). Looking at the profiles in Fig. 5.5(b), most of the search is contained near the initial feasible point, with very few exploratory actions. As a result, many iterations are not helpful in finding a better treatment. Meanwhile, the profiles in Fig. 5.5(c) demonstrate much more exploration of the design space $\mathcal{X}$, but with several constraint-violating queries. Using SEBO allows a middle-ground result. In Fig. 5.5(a), the observations exhibit more exploration compared to the standard safe BO, while still strictly adhering to the constraint.

[3]To implement SEBO, we modified components of Ax [123]. Ax interfaces with BoTorch [163] to perform BO. These tools were used with their default settings. Modifications to Ax are detailed in the codes available at `https://github.com/kchan45/SafeBOPlasma`.

[4]We note that $\tilde{\beta}$ is selected by the user, and $\beta = \tilde{\beta}\frac{\pi}{2}$ to match the implementation in Ax.

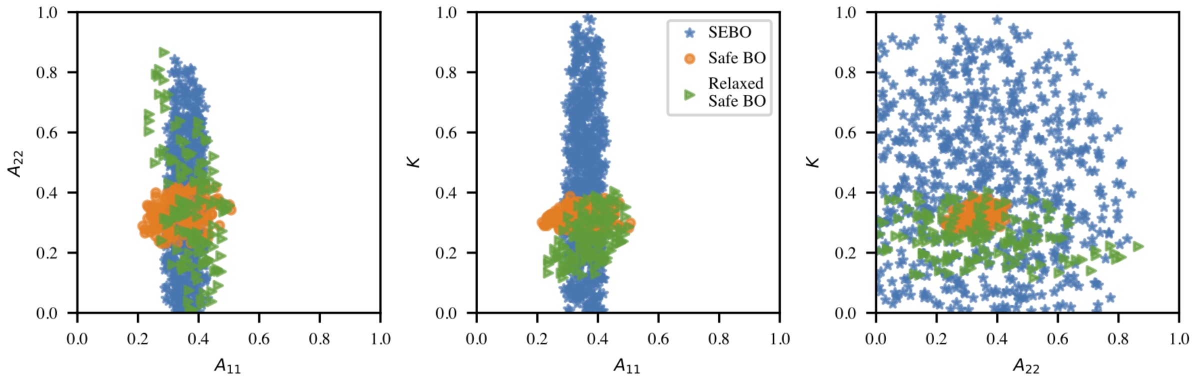


Figure 5.6: Comparison of revealed feasible sets of SEBO, safe BO, and the relaxed formulation of safe BO at iteration 30; the feasible set is denoted by $\hat{\mathcal{F}}_{30}$. Blue stars indicate $\hat{\mathcal{F}}_{30}$ of SEBO; orange circles indicate $\hat{\mathcal{F}}_{30}$ of standard safe BO; and green triangles indicate $\hat{\mathcal{F}}_{30}$ of the relaxed safe BO. While 5 parameters were included in the design space of BO, i.e., $x \in \mathbb{R}^5$, we show the revealed set for 3 of the parameters A_{11}, A_{22}, and K since they are deemed the most influential to the objective and constraints. Values of the parameters are normalized.

This result shows that SEBO uses and retains the information from the relaxed problem. We note that the standard safe BO appears to violate the constraint for two observations, and may be a result of a poor choice in β since the safety guarantees are probabilistic in nature.

Furthermore, we show that the revealed feasible set for SEBO is larger than what is estimated by standard safe BO in Fig. 5.6. By the end of 30 iterations, the posterior models can be used to visualize the revealed feasible set $\hat{\mathcal{F}}_{30}$. In Fig. 5.6, blue stars indicate $\hat{\mathcal{F}}_{30}$ of SEBO; orange circles indicate $\hat{\mathcal{F}}_{30}$ of standard safe BO; and green triangles indicate $\hat{\mathcal{F}}_{30}$ of the relaxed safe BO. We select a subset of parameters to examine and visualize, namely A_{11}, A_{22}, and K, due to their influence on the states s; A_{11} and A_{22} are the diagonal elements of A, and K is the exponential base of the CEM delivery. The subplots of Fig. 5.6 represent the 3 planes of the 3-dimensional space. Relaxed safe BO and SEBO both have larger $\hat{\mathcal{F}}_{30}$. In general, since the main influence of constraint violation involves temperature, A_{11} exhibits the most restrictive range of feasible parameters. Meanwhile, the influence of A_{22} and K are less influential with respect to safety. A_{22} relates to the total optical intensity, and K describes the rate of thermal dose delivery (CEM), both of which have no influence on safety. Nonetheless, standard safe BO provides a myopic view of the design space. $\hat{\mathcal{F}}_{30}$ of relaxed safe BO has a more holistic view of what is deemed feasible/unfeasible, so its region is larger than standard safe BO, but smaller than SEBO because it has explored more areas of infeasibility.

5.4 Conclusions

This chapter presented multi-objective BO (MOBO) as an effective strategy for the adaptation of deep learning-based control policies and safe explorative BO (SEBO) as a novel extension of penalty-based safe BO both in an effort towards personalized plasma medicine. Each method demonstrated how the direct manipulation of control policy parameters can be achieved and how, in a few iterations, adapt initial policies to an individual subject. In one direction, we showed how computationally expensive control policies (e.g., robust MPCs) can be cheaply approximated by deep neural networks (DNN). The DNN can be updated by modifying its parameters; we showed that MOBO can adapt a DNN policy in few learning iterations and determine trade-offs for plasma treatments on an individual. In a complementary direction, we demonstrated SEBO, which updated the parameters of an MPC policy in a safe, but explorative manner. Instead of remaining near a locally optimal point, SEBO safely manipulates control policies towards a more individualized global optimum. Each of these methods builds upon a framework based on BO and highlights aspects that are relevant to point-of-care personalized plasma medicine (i.e., control policies on resource-limited hardware and safety guarantees when exploring control policy design spaces). Immediate future directions can involve hardware-in-the-loop simulations and experiments, expanding upon the trade off between volume-of-improvement in the feasible space versus the performance improvement, and combining these ideas towards control-on-a-chip implementation.

Chapter 6

Integrating Learning and Cold Plasma Interactions to Identify and Differentiate Biological Materials[1]

Cold atmospheric plasmas (CAPs) affect biological materials via minimally-destructive chemical, thermal, and electrical interactions that are observable via common plasma characterization measurements. Further, for cases in which the interface to be characterized is already exposed (e.g., early skin cancer detection), CAPs can be used in a non-invasive manner for real-time identification. We leverage the sensitivity of CAP interactions with biological interfaces to identify and differentiate biological tissues by using real-time chemical (via optical emission spectra) and electrical (via voltage probes along the circuit) measurements. These information-rich measurements have embedded physics knowledge about the plasma chemistry and its interactions with biological tissues. Thus, we incorporate common physics knowledge to extract and analyze such measurements using machine learning. Finally, we demonstrate that biological tissues can be differentiated with high accuracy, and in proof-of-concept studies, we show that this novel sensing method can achieve up to 99% test accuracy when differentiating four tissue types (skin, muscle, bone, and fat) of an ex vivo *chicken model. As a result, there is potential for CAPs to augment the medical diagnostic toolkit, including in cancer detection, vascular studies, and real-time surgical analysis.*

6.1 Introduction

A major challenge in cancer diagnosis involves the early identification and differentiation between healthy and malignant tissue. The primary tool for cancerous cell determination is the histological evaluation (i.e., examination of the cell structures under microscope) of an

[1]This chapter was adapted with permission from the coauthors from [79].

extracted piece of tissue. This is an invasive procedure that involves the physical removal of potentially-afflicted tissue (via surgery) and is not done in real-time [174]. Particularly in the skin cancer realm, a non-invasive and real-time procedure to identify and discriminate cancerous tissues would significantly benefit patients. Creating one would allow for earlier diagnosis and could help avoid potentially unnecessary surgical procedures that can have complications and that are time consuming. A few alternatives have been proposed recently, but are not widely used, due to various limitations involving the need for expert analysis. One such example involves the use of high-frequency ultrasounds to obtain a high-resolution image of the skin structure via sound wave reflection measurements [175]. Another very recent example involves the use of optical coherence tomography (OCT) to perform a "virtual" biopsy. In this "virtual" biopsy, the procedure involves a non-invasive three dimensional scan of the tissue. The scan is used to generate a stained tissue image, similar to that of a true stained sample, which may be used instead of a true stained sample to provide diagnostic insights [176]. To replicate and potentially replace expert analysis for real-time and early diagnosis, more recently, there have been advances in artificial intelligence (AI) for medical image analysis. The accuracy of AI-based automatic detection and diagnostic systems has been shown to be comparable to that of experienced physicians in radiology [177, 178] with the turn-around time significantly improved, which can pose a potential avenue to speed up cancer diagnoses. In fact, medical image analysis using AI for skin cancer detection has been proposed in several studies [179, 180], but this method still lacks in reliability since it depends purely on visual appearance of the afflicted area. Thus, this method still relies on subjective characteristics of the macro-scale morphology of the tissue. Cancerous tissues, meanwhile, are known to have physiochemical properties that are significantly different from non-cancerous/healthy tissue.

Since the physiochemical properties of different tissue types play an important role in the existing gold-standard (i.e., histological analysis) toolkit, it must also be part of any new and innovative approach, and a few such properties have been explored. For example, Glickman et al. [181] proposed a method based on the measurement of electrical conductivity and capacitance of melanoma cells. Their technique required the use of disposable gold needles in a 8×8 matrix designed to penetrate the stratum corneum in the tested skin area and was calibrated based on surrounding healthy tissue. This technique showed high sensitivity (92% compared to 75% for physicians), but low specificity (67% compared to 87% for physicians) for the diagnosis of melanoma [181]. Another method by Spether et al. [174] and Büger et al. [182] relied on analyzing the chemical composition of cancerous tissue. This method used optical emission spectroscopy of an atmospheric thermal plasma (with temperatures that exceed 1000°C) generated using an electrosurgical tool during a tumor resection. The tissue in contact with the plasma is vaporized allowing for the excitation of the molecular makeup of the tissue, which emits characteristic optical emission spectra that can discriminate cancerous versus non-cancerous tissue. This method reported an accuracy of 95% in differentiating between cancerous and healthy liver tissues [182]. However, it is destructive and not applicable in skin cancer detection and early diagnosis. Finally, tumor cells have been reported to present lower thermal conductivity compared to healthy cells [183], yet no known cancer detection system based on thermal conductivity has been proposed. To summarize, key limitations to each of the above methods include the focus on a single property, the lack of specificity, and/or the destructive nature of the examination.

In this work, we investigate a new approach to real-time tissue differentiation using cold atmospheric plasmas (CAPs) to provide insight into various physiochemical properties of the biological tissue. CAPs are a form of partially-ionized gaseous matter that can be generated at near-room temperature and atmospheric pressure, which allows for a "gentle" interaction with biological materials, such as tissues. This "gentle" interaction means that CAPs can be used in a non-invasive, minimally-destructive manner. Thus, CAPs have sparked the development of an entirely new field that lies at the intersection of (non-equilibrium) plasma and medicine. Plasma medicine has now grown into a field that ranges in proposed treatments from disinfection to wound healing to cancer therapies [16, 17]. The key to CAPs' success in medicine has been the plasma-surface interactions with biological systems, which can be evaluated by monitoring the physiochemical properties of those interactions. However, plasma medicine hinges on expertise in plasma physics and chemistry, as well as in the biochemistry of the plasma-interface interactions [16]. AI, particularly machine learning (ML), can play a crucial role in bridging that gap between elucidating the underlying physics of plasma-interface interactions in real-time [19]. Recently, there have been some reports on learning-based control for CAPs [19, 63, 76, 145], on ML for predicting biological outcomes of CAP treatments [184], and on predicting physiochemical properties to differentiate (non-biological) materials [33, 185–187]. However, no known works (to the authors' best knowledge) use CAPs to differentiate biological targets.

This work aims to use a combination of CAP effects and ML to detect differences in, differentiate between, and diagnose/identify biological tissues. We investigate how CAP interactions with biological tissues can be combined with ML to identify biological tissues in a real-time, non-invasive manner. To do so, we developed and used an all-in-one CAP generation and CAP effect measurement device. The CAP device is based on a prior configuration [4] commonly proposed for plasma medicine combined with an automated data acquisition structure using real-time capable measurement devices commonly used in plasma (and plasma-surface) characterization [63]. This setup was used to collect chemical and electrical data on *ex vivo* chicken leg models at various points consisting of different tissue types (i.e., skin, muscle, bone, and fat). The chemical data primarily consisted of optical emission spectra, and the electrical data consisted of electrical waveforms associated with the plasma-target interactions. Once data were collected with the setup, physics- and biologically-informed data analysis and processing were used to identify and select features of the data to be used in training ML models. This data processing included (but is not limited to) peak selection (to identify differences in chemical properties) and physical transformation of the data to understand underlying physics. After this physics-informed data processing, we evaluated a variety of supervised learning classification techniques to achieve high test accuracy across all models. To this end, this work aims to provide a viable avenue towards a novel (skin) cancer detection technology that lies at the intersection of physics, mathematics, and biology.

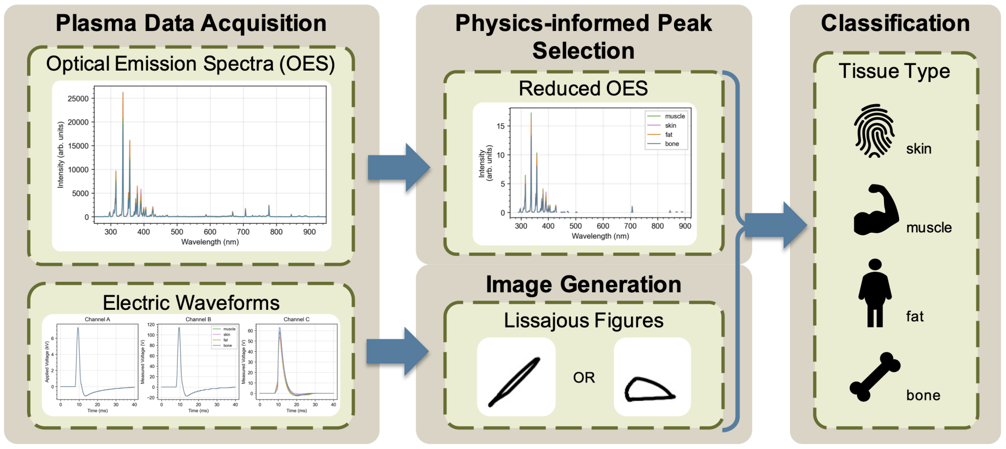


Figure 6.1: Information flow of our proposed tissue diagnostic workflow. We start with an automatic data acquisition setup using the plasma gun setup as described in Figure 2.3. The raw data that is collected is in the form of optical emission spectra (OES) from the spectrometer and electric waveforms from different locations and recorded by an oscilloscope. The data is transformed and reduced in dimension in physics-informed ways: the OES are manually reduced to important peaks and the electric waveforms are converted to Lissajous figures. A classification technique for biological tissue detection and identification is trained using labeled data and is tested on unseen data.

6.2 Methods

6.2.1 Data Collection

The device configuration used in this chapter was detailed in Chapter 2.3. Using the Plasma Gun setup, we generated a dataset of $11,456$ samples collected at 100 time intervals over seven chicken leg models at four different tissues types at multiple locations. Note that some data were manually excluded when the system did not ignite the gas into a plasma or acquire the data.

An additional analysis of the plasma was done using a high resolution optical emission spectra (OES). We used a 0.320 m focal length spectrometer (IsoPlane SCT 320) coupled with an ICCD camera (PiMax4 by Princeton Instruments). Here, the ICCD camera was synchronized with the high voltage pulse of the CAP device and the total integration time was 10 μs per 5000 on-chip accumulations. Low resolution OES were used in the data-driven classification, and the high resolution OES were used to validate physical findings between the spectral differences observed in the low resolution data.

6.2.2 Classification Methods for Differentiation

Classification is a category of supervised ML wherein a model predicts the “label” of given input data. In this work, this reflects the correct prediction of the type of biological chicken model tissue based on chemical and electrical input data. Figure 6.1 illustrates the information flow of the proposed classification technique. Raw chemical data (in the form of OES) and electrical data are captured and saved from the Plasma Gun. The data are processed in a second step, where peaks of the OES that indicated plasma-biological significance were selected, and the remainder of the spectra were discarded and electric waveforms were processed into Lissajous figures. Each of these types of processed data were labeled with the tissue type of the sample (skin, muscle, bone, and fat) and used to train and evaluate various ML models. Once a model is trained, the first two steps can be re-used during real-time inference to generate a prediction of the tissue type as illustrated in the final step of Figure 6.1.

In this work, we investigated the value of each type of data (only chemical, only electrical, or a combination of chemical and electrical) for tissue identification by training ML models for each type of model input. Furthermore, we compared the performance of several types of ML models for multiclass classification, including k nearest neighbor, decision trees, random forests, and neural networks. Each of k nearest neighbor, decision trees, and random forests was created and trained using the `scikit-learn` [188]. Fully-connected neural networks were created and trained using `Tensorflow` [189].

6.3 Results

6.3.1 Qualitative Differences between Chicken Model Tissues

Optical Emission Spectra

OES are commonly used in plasma diagnostics, however, this paper reports one of the first uses of this information for the characterization of the plasma-(bio)interface interactions. Figure 6.2 illustrates an exemplary OES of the helium plasma impinging upon different tissues of a chicken leg model. Using the automatic data acquisition setup, we obtained sample OES of the CAP interactions with skin, muscle, bone, and fat tissue of *ex vivo* chicken leg models over a treatment time of 50 seconds at 0.5-second sampling intervals. The helium CAP shows characteristic peaks for helium excitation at 587.6 nm, 667.8 nm, 706.5 nm, and 728.1 nm. Further, oxygen and nitrogen species typical of CAP operating in ambient air include O_2 (337.0 nm), O_3 (313.74 and 317.16 nm), N_2 (315.93, 337.13, 357.69, and 380.49 nm), NH (336.01 nm), NO (337.64 and 357.24 nm), HNO_2 (354.25 nm), N_2^+ (358.21, 391.44, and 427.81 nm), N_2O^+ (355.84 nm) and OH^+ (356.5 nm). Noticeable differences among the relative intensity of these peaks were observed for CAP interacting with different tissues, suggesting that the plasma chemistry can be influenced through interactions with different biological interfaces.

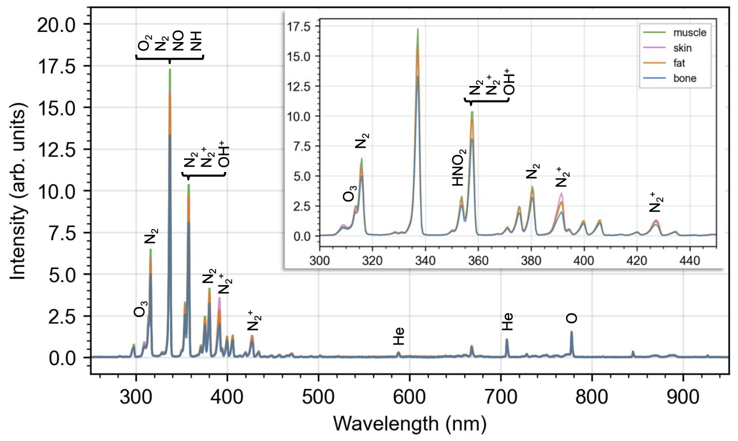


Figure 6.2: Exemplary optical emission spectra of helium plasma impinging upon skin, muscle, bone, and fat tissue of a chicken leg model. OES are collected over 50 seconds at 0.5-second sampling intervals and averaged. OES are normalized with respect to the He706 peak. Here, the spectra show distinct characteristics between peak heights that can be exploited in classification and/or clustering techniques.

Additional observed peaks that are associated with elements from the tissue are: Ca^{++} (373.69 and 393.29 nm), Na (589.6 nm). Furthermore, differences corresponding to peaks that can be associated with molecules from the tissue were also observed: Tryptophane (345 and 419.8 nm), Pepsin (380 nm), Collagenase (420 nm), Oxyhemoglobin (412 nm), Lactic acid (434 nm), Elastine (500 nm). We note that the location of these biologically-relevant peaks in OES are quite limited as there are few methods to obtain these spectra without degradation (i.e., with light induced fluorescence). Moreover, the emission from other elements often overlaps and obscures these peaks. To verify the existence of these biologically-relevant peaks, we used high resolution OES where data were collected at the plasma-tissue incidence point and at just the plasma plume itself. Comparing the two collection points allowed us to verify the existence of these peaks and use them to identify important features to feed into the ML models.

Electrical Characteristics - Lissajous Figures

Lissajous figures are visualizations of a system of parametric equations, typically of two waveforms. In the case of CAP systems, charge-voltage (Q-V) Lissajous figures can be generated from measurements of the applied voltage and the charge deposited to target and electrodes. Charge is measured via the voltage drop through a capacitor. In creating such figures for the CAP system, physics information is encoded into a visual representation of the data, in particular, the area enclosed by the Q-V figure is the amount of energy deposited

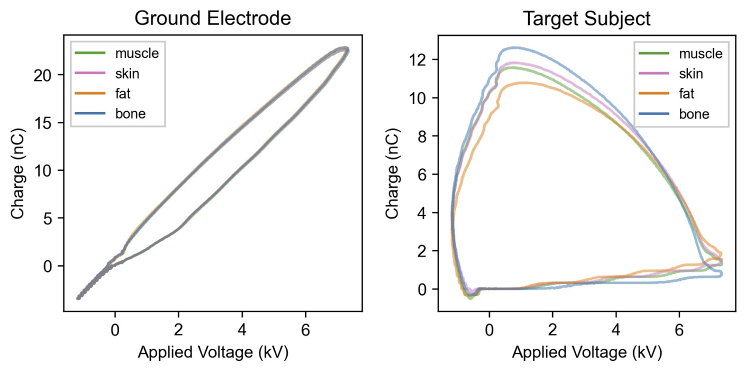


Figure 6.3: Exemplary charge-voltage figures of helium plasma impinging upon skin, muscle, bone, and fat tissue of a chicken leg model. Electrical waveforms are collected over 50 seconds at 0.5-second sampling intervals and averaged. The left figure illustrates the charge collected at the ground electrode versus the applied power over 50 seconds, and the right figure illustrates the charge collected at the target subject versus the applied voltage. The left figure shows overlapping electrical characteristics and illustrates that the generated plasma is of consistent quality. The right figure illustrates the different shapes of these electrical characteristics between different tissue types, which can be exploited in differentiating or identifying biological tissues.

onto the target electrode.

Figure 6.3 illustrates exemplary Q-V figures to demonstrate the electrical properties of the plasma-tissue interactions. Figure 6.3 illustrates the Q-V figures of measurements of the CAP system at two locations (see Figure 2.3, pentagon markers): one probe located at the outer grounded electrode of the CAP device and one probe connected in series with the biological tissue after the compensation circuit. Qualitatively, it is possible to observe how the Q-V plot for the target is clearly affected by the type of tissue. Contrarily, the Q-V plot for the ground electrode remains essentially constant. This is because the internal configuration of the CAP device, where the plasma ignites, is fixed. We can assume that the plasma ignition inside the device is not significantly affected by the presence of the tissue compared to the plasma's propagation and interactions with the target tissues. The variation of the Q-V figure for the target can be exploited in differentiating or identifying biological tissues. We note that there was a similar magnitude of variance between samples of the same tissue type that can have consequences on the classification capability and is discussed further in the subsequent section.

Classification Accuracy

Following the data processing steps outlined in Figure 6.1, we trained several ML models to classify four different tissue types (skin, muscle, bone, and fat) of *ex vivo* chicken leg

	Test Accuracy (%)		
Model	Average	Max	Min
Decision Tree	92.81 ± 0.71	94.02	91.79
Random Forest	98.65 ± 0.24	99.08	98.30
k Nearest Neighbor	98.21 ± 0.31	98.65	97.73
Neural Network	99.41 ± 0.25	99.69	98.95

Table 6.1: Test accuracy (mean $\pm$ standard deviation) of various ML models trained on combined chemical and electrical data. Statistics are established over 9 random resamplings of the train/validation/test datasets and initializations of the models.

models. Data were split with an 80/20 training/test split where 20% of the total samples were reserved as unseen data for evaluation. The remaining 80% was split into a 92/8 training/validation split, where 8% of the training data would be reserved for validation. Each of the decision tree (DT), random forest (RF), and k nearest neighbor (kNN) was trained using the `fit` function, and optimal hyperparameters (max tree depth and number of neighbors) were selected via a 5-fold cross-validation score. The neural network (NN) was constructed as a fully-connected network with three hidden layers, 100 nodes per layer, a batch normalization layer after the input, and a dropout layer (with 40% dropout) prior to the output layer. The NN was trained using a batch size of 128 over 30 epochs, and the best model was selected according to the best validation loss. Table 6.1 reports the test accuracy of these various ML models trained on a combination of the chemical (OES) and electrical (Q-V image of the target) data. Accuracies are reported as the mean and confidence bounds (standard deviation) over 9 random initializations of each model. All models show high discriminative capabilities ($> 90\%$ test accuracy on average).

Figure 6.4 shows examples of confusion matrices of classifiers trained on different sets of chicken model data. A confusion matrix is a common visual representation to illustrate the accuracy of a classification model. In a confusion matrix, the model predictions are plotted along one axis, while the true labels are plotted along the orthogonal axis. This results in a $c \times c$ grid where c is the number of classes, and the diagonal indicates when the prediction is equivalent to the truth. Hence, higher values along the diagonal of the confusion matrix indicate a higher accuracy. Further, the confusion matrix provides a representation of what "confuses" the model, i.e., what the model's incorrect predictions should be in truth. Specifically, Figure 6.4 illustrates the predictive capability of a DT and a NN using chemical-only data, electrical-only data, and a combination of both. In general, using chemical data results in higher tissue classification accuracy as indicated by most sample predictions lying on the diagonal in both types of models when chemical data is included. Furthermore, a lower complexity model (e.g., a DT) can experience a performance boost (more samples along the diagonal) when incorporating additional data compared to a higher complexity

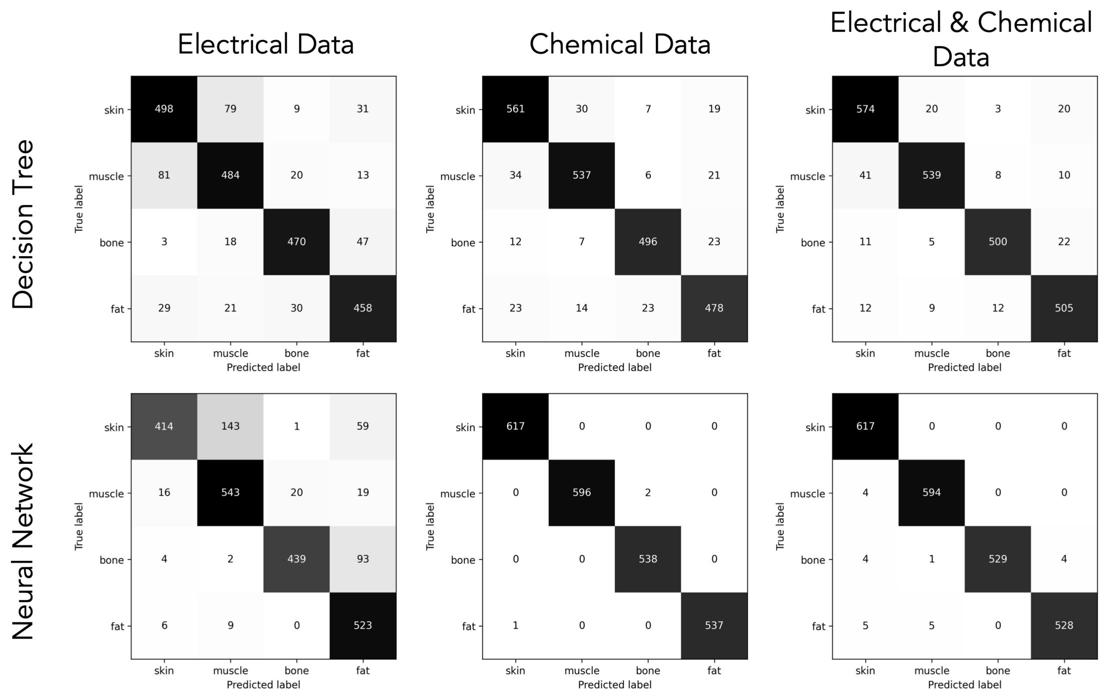


Figure 6.4: Confusion matrices of a decision tree and a neural network trained different datasets (electrical data only, chemical data only, and a combination of chemical and electrical data).

model (e.g., a NN). This performance boost is illustrated in Figure 6.4 when the DT (top row) increases in total number of samples along the diagonal (i.e., accuracy): 1910 (83.4%) for the electrical data versus 2072 (90.4%) for the chemical data versus 2118 (92.4%) for the combined electrical and chemical data). The NN does not necessarily experience this performance boost, particularly when comparing chemical data versus combined data: 1919 (83.8%) for the electrical data versus 2288 (99.9%) for the chemical data versus 2268 (99.0%) for the combined electrical and chemical data.

6.4 Discussion

In this work, we demonstrated the potential for CAPs as a diagnostic tool for the identification and classification of biological tissues. CAPs exhibit distinct chemical and electrical characteristics when subjected to different biological materials. In particular, OES, which capture the gas-phase chemical reactivity of the plasma at the plasma-tissue incidence point, illustrates distinctions in the reaction/activation potentials of both plasma and biological species. In particular, certain peaks of the OES can correspond to the excitation of com-

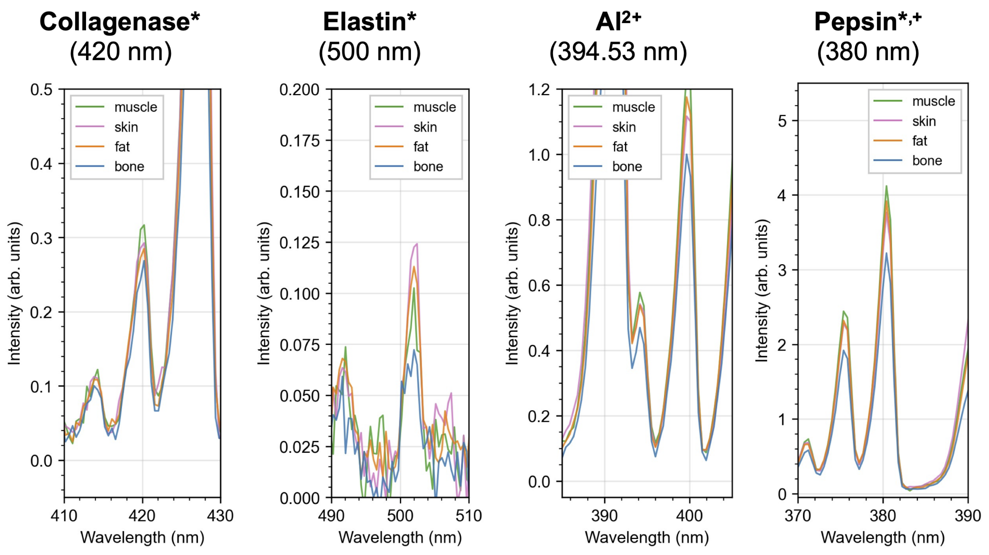


Figure 6.5: Selected biologically-relevant peaks [7, 8] from the optical emission spectra (normalized against the He706 peak) of cold-atmospheric plasma-treated chicken models. *Elastin represents elastin cross-links, and collagenase and pepsin represent collagenase-digestible and pepsin-digestible collagen cross-links. +Further, the pepsin-digestible collagen peak is likely heavily overlapping with the excited nitrogen peak (380.5 nm).

pounds found in different amounts in each tissue type. In Figure 6.5, we highlight a few of these compounds: collagenase-digestible collagen cross-links, elastin cross-links, aluminum, and pepsin-digestible collagen cross-links [7]. Collagen is the main structural protein found in various connective tissues and can be broken down via enzymes such as collagenase and pepsin. Presence and distribution of these forms of collagen cross-links vary with each tissue type, and the OES can capture this difference from the variation in peak height when normalized with a non-biological peak (i.e., He706). Elastin is another protein that is commonly found in connective tissues, allowing for tissues to revert back to their original shape (i.e., enabling elasticity for tissues). Intuitively, elastin is prevalent in skin, muscle, and even fat tissues, whereas its prevalence in bone tissue is less. This trend is reflected in Figure 6.5 for elastin as the bone peak is low compared to the other three tissue categories. Finally, aluminum is a common element found in biological tissues as a result of environmental factors, including diet. On one hand, permeation of aluminum can differ between tissue types, allowing us to use aluminum peak(s) to distinguish between biological materials. From a broader perspective, the difference in aluminum can be used to infer deviations from the normal and/or aid in diagnosis of conditions such as skin cancer [190]. These peaks (and others not listed here) from the low resolution OES captured the distinctions of different tissues such that the ML models were able to classify (with high accuracy) different tissue

	Average Test Accuracy (%)		
Model	Electrical Data	Chemical Data	Electrical & Chemical Data
Decision Tree	84.31 ± 0.74	91.01 ± 0.57	92.81 ± 0.71
Random Forest	89.55 ± 0.65	$\mathbf{99.02 \pm 0.25}$	98.65 ± 0.24
k Nearest Neighbor	$\mathbf{91.85 \pm 0.47}$	98.92 ± 0.24	98.21 ± 0.31
Neural Network	82.06 ± 3.38	98.90 ± 0.90	$\mathbf{99.41 \pm 0.25}$

Table 6.2: Test accuracy of various ML models trained on different datasets. Statistics (mean ± standard deviation) are reported across 9 random resamplings of the train/validation/test datasets and initializations of the models.

types.

Electrical measurements also demonstrate a level of distinction between different tissue types, as seen in the variance of shapes in Figure 6.3 and the mid-high accuracy results summarized in Table 6.2. Dielectric properties of various tissue types have been explored extensively and show some distinctions between different tissues [191]. In this work, we found no significant improvement in the classification capability of classifiers when trained with chemical-only data versus with combined chemical and electrical data, which indicates that the chemical data likely provides the bulk of the discriminative qualities of plasma-tissue interactions. Despite this, there remains significant potential in the use of electrical measurements in data-driven identification and differentiation of biological tissues, as the form and manipulation of the electrical waveforms can be improved to higher resolution images, modified to emphasize certain characteristics, or augmented with additional sensor data (including current). Overall, this investigation of various machine learning classification methods demonstrated reliable estimation of the different tissue types. All models demonstrated strong classification capability of the input OES and electrical image data with random forests and neural networks demonstrating near-perfect classification accuracy.

6.5 Conclusions

This study introduced a new method for real-time tissue identification using cold atmospheric plasmas and ML. We collected chemical (optical emission spectra) and electrical (circuit analysis) data from minimally-destructive plasma interactions with *ex vivo* chicken leg models. Data were captured for distinct tissue types and were shown to be discriminative across tissue types, which are likely due to distinct physiochemical properties that are exposed when different tissue types interact with the CAP. As such, various ML models, were able to classify tissue types of *ex vivo* chicken leg models from features including the chemical and electrical data with at least 90% test accuracy on average and up to 99.5% test accuracy for the best model. We identified and transformed the data into physically- and biologically-relevant features and elaborated on several components that may have led to such high predictive power. The physics-informed discoveries in this work leave significant potential for future efforts in modeling CAP-(bio)interface interactions and provide a means towards a real-time non-invasive skin cancer detection tool. For this type of data, additional avenues of exploration can involve convolutional neural networks to better extract the features of the electrical image data, nonlinear dimensionality reduction strategies to directly extract important features, and unsupervised learning techniques to cluster the data into distinct groups.

6.6 Additional information

Code and data used in this study are available at `https://github.com/Mesbah-Lab-UCB/CAP-Sensor4Bio`.

Chapter 7

Conclusions

This dissertation investigated the end-to-end design of embedded control systems that enable individualized plasma treatment regimens in plasma medicine. The results of this work indicate that data-driven optimization is a versatile framework to design and adapt operational parameters of plasma treatments to create safe and efficacious treatment regimens. This chapter summarizes major conclusions from each chapter and presents a few open research questions to be addressed going forward.

7.1 Summary

This dissertation investigated two key challenges towards enabling safe and effective point-of-care devices for precision plasma medicine. Challenges associated with the lack of knowledge between the digital control policy design, the computing hardware design, and the plasma-(bio)interface informed the choice and development of a Bayesian optimization (BO)-based framework to design an end-to-end embedded control system for point-of-care plasma devices. While this dissertation had a strong focus on plasma biomedical devices, this end-to-end design framework for embedded control systems is broadly applicable to arbitrary choices in control policies, computing hardware, and system specifications. Challenges related to the incomplete knowledge of plasma-(bio)interface dynamic interactions and variability between plasma device operation or between patients/individual interfaces motivated the use of BO (and the extension of safe BO) to adapt CAP treatment protocols over multiple treatments.

The first part of this dissertation (Chapters 2 and 3) focused on the particular CAPJ testbeds used and provided problem formulations of CAP treatments considered in this work. CAPJs were the focus of this work due to their widespread use in plasma medicine owing to their versatility and portability. Each of the CAPJ testbeds was outfitted with automated data acquisition and actuation that enabled operational control to be applied. Further, a code framework was developed to enable run-to-run control over operational outcomes. Intermediate predictive control policies were described in Section 3.2 and the run-to-run framework in general was described in Section 3.3. The formulation of these control problems gave rise to a structured approach to address adaptive plasma treatments. The versatility of this framework was illustrated in various aspects of the design pipeline.

In Chapter 4, we evaluated the framework through a broad lens relating the control policy program (digital/software) design to the physical computing hardware design. Chapter 4 created a unifying template for hardware-software co-design that was designated as control-on-a-chip (CoC) design. Deep learning (DL) was key to the unified CoC design template as DL provides a streamlined connection between hardware and software. Then, BO provided an optimization framework that accounted for the multi-objective nature of the control design problem and categorical design space inherent to hardware design choices. Through closed-loop simulations and experiments, we verified that multi-objective BO (MOBO) systematically determined trade-offs in the CoC design process and resulted in an adequate estimation of the Pareto frontier. As a result, Chapter 4 illustrated the versatility of (MO)BO to design arbitrary control policies on arbitrary hardware, enabling complex embedded control systems that are necessary for plasma control at the edge.

In Chapter 5, we explored the feasibility of individualized plasma treatment regimens by illustrating modifications to BO to address two aspects related to plasma medicine, namely adaptive DL-based control policies for embedded control systems and safe exploration of control policy design space. Previously, in Chapter 4, we investigated the architecture of DL (i.e., the structure of the control policy), which is useful in designing an initial feasible control policy. Different from Chapter 4, Section 5.2 directly modified the parameters of the DL-based control policy, which is useful when adapting control policies to individuals. The direct manipulation of control policy parameters is the subject of the larger field of policy adaptation in reinforcement learning (RL), but in general, these methods can be ill-suited to the limitations of CAP systems (e.g., in balancing multiple objectives and in safe policy exploration). In one direction, we demonstrated that MOBO systematically determined trade-offs in the direct adaptation of a DL-based control policy and resulted in quantification of the trade-off between constraint satisfaction (i.e., safe operation) and control performance (i.e., treatment speed). In a complementary direction, we demonstrated that individualized control policies can be determined in a safe manner (i.e., without violating safety-critical constraints) using a novel BO strategy, named Safe Explorative BO (SEBO), that incorporates the volume of improvement in the feasible/safe set. SEBO was capable of much more exploration of the design space compared to an existing penalty-based safe BO method, which ensured that recommended designs do not get stuck in a locally feasible space.

In Chapter 6, we took a slight deviation from the adaptive control policy framework, and instead, explored a novel application of CAPs for biomedical use: bio-interface characterization. CAPs are uniquely capable of producing minimally-destructive effects during its interaction with biological tissues. We demonstrated that real-time chemical and electrical measurements of CAP interactions with biological tissues can be processed in physics-informed ways and fed into supervised machine learning (ML) models to identify and differentiate the type of biological tissue. We trained ML models that were capable of up to 99% test accuracy when predicting tissue types from real-time data.

The key outcome of this dissertation revolves around an end-to-end perspective of control on the edge for precision plasma medicine. Point-of-care devices and control on the edge dictate the need for embedded control systems. Diverse patient profiles and variation among plasma-(bio)interface interactions necessitate adaptive and individualized CAP treatment regimens. A data-driven optimization perspective has been shown to address each of these needs to produce feasible embedded control systems on point-of-care devices that are used to personalize CAP treatments. While the context of this work was in plasma medicine, the versatile structure of data-driven optimization allows it to be broadly applicable to other plasma processing systems, including semiconductor manufacturing [192]. The next section outlines some open research questions related to this body of work.

7.2 Recommendations for Future Work

7.2.1 Control Policy Design in the Context of Privacy and Security

Portable medical devices and health monitors are becoming pervasive throughout society. A major challenge in the era of computing involves the increasing vulnerability of computational systems to adversarial attacks. Medical devices, in particular, should be designed with built-in features of privacy and security, which can be established through software, hardware, or wireless communication, to protect patient data. While significant efforts have been made to counteract potential vulnerabilities, most current methods rely on detection and damage mitigation rather than adaptive and preemptive action [193]. An extension of the data-driven optimization framework could incorporate the networked communication design between edge devices edge servers. This communication design can have impacts on the rest of the design pipeline, which makes the overall design difficult to understand. Additionally, several of the challenges involved in designing control policies for Internet-of-(Medical)-Things (IoMT) include expensive-to-evaluate systems, time constraints to mitigate damage or exposure, and uncertainty in the face of constantly-evolving computing systems [194,195].

7.2.2 Embedded Physics-informed Data-driven Learning

Machine learning has been a key driver to some of the latest advances in low temperature plasma systems and in its control [196,197]. However, general data-driven approaches are not physically interpretable and do not typically generalize or extrapolate beyond the patterns existing in the training data. Recently, and with success in some fields, new architectures of learning that provide an explainable transformation of the input data have made significant breakthroughs in learning and understanding the underlying dynamics [198–200]. These models can then be used in a generative modeling context. A feature of generative modeling is that it aims to understand underlying patterns for prediction and provide an uncertainty estimation of those predictions [197]. The investigation of more physics-informed machine learning architectures and generative modeling holds promise for more interpretable, and likely more accurate, predictions about the plasma-biological interactions that will aid in control and optimization within plasma medicine. One example may be with graph neural networks (GNNs) in modeling chemical reaction pathways of plasma systems [199]. Graphs can be a natural way to represent complex (plasma) reaction networks [201], and then GNNs are a learning strategy that employs convolutions to can pass information between nodes of the graph [202]. Embedding GNNs within control strategies can provide an interpretable evolution of (plasma) chemistry that have hardware-compatible computational structures. Other examples can include neural ordinary differential equations (ODEs) [203] or LyaNet [204], which embed dynamics in the form of ODEs or a control-theoretic strategy to train ML models, respectively.

7.2.3 Expansion Towards Preclinical and Clinical Studies

Clinical trials for CAP treatments are increasing as the therapeutic benefits of CAPs in plasma-biological systems are gaining traction. Since, Isbary *et al.* in 2010 published one of the first clinical successes of CAP treatments in medicine [205], there have been a number of clinical studies and development of CAP devices [206]. Meanwhile, data-driven optimization has been used in few studies to optimize, adapt, and/or personalize medical treatments [207, 208]. An important next step in enabling CAPs for medical treatments will involve demonstrations that these data-driven methods work in medical practice. This may involve preliminary *in vitro* and then *in vivo* studies to validate the data-driven approach on a biological system.

7.2.4 Plasma-enabled Sensing of (Biological) Characteristics

As previously shown for non-biological systems [33] and as shown in Chapter 6, CAPs hold promise in their capability to identify and distinguish between different material surfaces/interfaces. In this direction, generative modeling can further expand the capabilities of what has been done in this dissertation. Because of CAPs' minimally-destructive nature, the use of CAP-based soft sensing holds promise for a variety of applications ranging from battery manufacturing to semiconductor fabrication to (early) detection of malignant tissues.

Bibliography

[1] D. Gidon, D. B. Graves, and A. Mesbah, "Effective dose delivery in atmospheric pressure plasma jets for plasma medicine: A model predictive control approach," *Plasma Sources Science and Technology*, vol. 26, no. 8, p. 085005, 2017.

[2] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos, and A. Mesbah, "Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks," *Computers & Chemical Engineering*, vol. 145, p. 107174, 2021.

[3] L. Lin and M. Keidar, "A map of control for cold atmospheric plasma jets: From physical mechanisms to optimizations," *Applied Physics Reviews*, vol. 8, no. 1, 2021.

[4] A. Stancampiano, T.-H. Chung, S. Dozias, J.-M. Pouvesle, L. M. Mir, and E. Robert, "Mimicking of human body electrical characteristic for easier translation of plasma biomedical studies to clinical applications," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 4, no. 3, pp. 335–342, 2019.

[5] D. Gidon, *Advanced Control of Atmospheric Pressure Plasma Jets for Medical Applications*. PhD thesis, University of California, Berkeley, 2019.

[6] A. D. Bonzanini, *Safe and Fast Learning-based Model Predictive Control of Nonlinear Systems with Applications to Cold Atmospheric Plasmas*. PhD thesis, University of California, Berkeley, 2022.

[7] N. Kollias and G. N. Stamatas, "Optical non-invasive approaches to diagnosis of skin diseases," *Journal of Investigative Dermatology Symposium Proceedings*, vol. 7, no. 1, pp. 64–75, 2002.

[8] M. J. Myers, J. D. Myers, B. Guo, C. Yang, C. R. Hardy, J. A. Myers, A. G. Myers, and S. M. Christian, "Non-invasive in-situ detection of malignant skin tissue and other abnormalities using portable LIBS system with fiber spectrometer and eye-safe erbium glass laser," in *Optical Diagnostics and Sensing VIII*, vol. 6863, pp. 217–226, SPIE, 2008.

[9] T. Philbeck and N. Davis, "The fourth industrial revolution," *Journal of International Affairs*, vol. 72, no. 1, pp. 17–22, 2018.

[10] K. Schwab, *The fourth industrial revolution.* Crown Publishing Group, 2017.

[11] U. Rosolia, X. Zhang, and F. Borrelli, "Data-driven predictive control for autonomous systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 259–286, 2018.

[12] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.

[13] A. Mesbah, K. P. Wabersich, A. P. Schoellig, M. N. Zeilinger, S. Lucia, T. A. Badgwell, and J. A. Paulson, "Fusion of machine learning and MPC under uncertainty: What advances are on the horizon?," in *Proceedings of the American Control Conference*, pp. 342–357, 2022.

[14] M. G. Kong, G. Kroesen, G. Morfill, T. Nosenko, T. Shimizu, J. Van Dijk, and J. Zimmermann, "Plasma medicine: an introductory review," *New Journal of Physics*, vol. 11, no. 11, p. 115012, 2009.

[15] K. Weltmann and T. Von Woedtke, "Plasma medicine – current state of research and medical application," *Plasma Physics and Controlled Fusion*, vol. 59, no. 1, p. 014031, 2016.

[16] M. Laroussi, "Plasma medicine: a brief introduction," *Plasma*, vol. 1, no. 1, pp. 47–60, 2018.

[17] T. Bernhardt, M. L. Semmler, M. Schäfer, S. Bekeschus, S. Emmert, L. Boeckmann, *et al.*, "Plasma medicine: Applications of cold atmospheric pressure plasma in dermatology," *Oxidative medicine and cellular longevity*, vol. 2019, 2019.

[18] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial Intelligence in Medicine*, vol. 104, p. 101822, 2020.

[19] A. D. Bonzanini, K. Shao, A. Stancampiano, D. B. Graves, and A. Mesbah, "Perspectives on machine learning-assisted plasma medicine: Toward automated plasma treatment," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 6, no. 1, pp. 16–32, 2021.

[20] S. Razdan and S. Sharma, "Internet of medical things (iomt): Overview, emerging technologies, and case studies," *IETE technical review*, vol. 39, no. 4, pp. 775–788, 2022.

[21] L. Sun, X. Jiang, H. Ren, and Y. Guo, "Edge-cloud computing and artificial intelligence in internet of medical things: architecture, technology and application," *IEEE Access*, vol. 8, pp. 101079–101092, 2020.

[22] D. C. Klonoff, “Fog computing and edge computing architectures for processing data from diabetes devices connected to the medical internet of things,” *Journal of Diabetes Science and Technology*, vol. 11, no. 4, pp. 647–652, 2017.

[23] M. A. Rahman and M. S. Hossain, “An internet-of-medical-things-enabled edge computing framework for tackling COVID-19,” *IEEE Internet of Things Journal*, vol. 8, no. 21, pp. 15847–15854, 2021.

[24] I. S. Chan and G. S. Ginsburg, “Personalized medicine: progress and promise,” *Annual review of genomics and human genetics*, vol. 12, pp. 217–244, 2011.

[25] M. A. Hamburg and F. S. Collins, “The path to personalized medicine,” *New England Journal of Medicine*, vol. 363, no. 4, pp. 301–304, 2010.

[26] M. R. Kosorok and E. B. Laber, “Precision medicine,” *Annual Review of Statistics and its Application*, vol. 6, pp. 263–286, 2019.

[27] J. B. Rawlings, “Tutorial overview of model predictive control,” *IEEE Control Systems Magazine*, vol. 20, no. 3, pp. 38–52, 2000.

[28] D. Q. Mayne, M. M. Seron, and S. Raković, “Robust model predictive control of constrained linear systems with bounded disturbances,” *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[29] A. Mesbah, “Stochastic model predictive control: An overview and perspectives for future research,” *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 30–44, 2016.

[30] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “GPU computing,” *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[31] I. Kuon, R. Tessier, J. Rose, *et al.*, “FPGA architecture: Survey and challenges,” *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[32] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, “High-level synthesis for FPGAs: From prototyping to deployment,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[33] D. Gidon, X. Pei, A. D. Bonzanini, D. B. Graves, and A. Mesbah, “Machine learning for real-time diagnostics of cold atmospheric plasma sources,” *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 3, no. 5, pp. 597–605, 2019.

[34] A. M. Hirst, F. M. Frame, M. Arya, N. J. Maitland, and D. O’Connell, “Low temperature plasmas as emerging cancer therapeutics: the state of play and thoughts for the future,” *Tumor Biology*, vol. 37, pp. 7021–7031, 2016.

[35] I. Adamovich, S. Baalrud, A. Bogaerts, P. Bruggeman, M. Cappelli, V. Colombo, U. Czarnetzki, U. Ebert, J. G. Eden, P. Favia, *et al.*, "The 2017 Plasma Roadmap: Low temperature plasma science and technology," *Journal of Physics D: Applied Physics*, vol. 50, no. 32, p. 323001, 2017.

[36] I. Adamovich, S. Agarwal, E. Ahedo, L. L. Alves, S. Baalrud, N. Babaeva, A. Bogaerts, A. Bourdon, P. Bruggeman, C. Canal, *et al.*, "The 2022 plasma roadmap: Low temperature plasma science and technology," *Journal of Physics D: Applied Physics*, vol. 55, no. 37, p. 373001, 2022.

[37] P. J. Bruggeman, F. Iza, and R. Brandenburg, "Foundations of atmospheric pressure non-equilibrium plasmas," *Plasma Sources Science and Technology*, vol. 26, no. 12, p. 123002, 2017.

[38] B. Denis, S. Steves, E. Semmler, N. Bibinov, W. Novak, and P. Awakowicz, "Plasma sterilization of pharmaceutical products: from basics to production," *Plasma Processes and Polymers*, vol. 9, no. 6, pp. 619–629, 2012.

[39] M. El Shaer, M. Mobasher, and A. Zaki, "Low-cost dielectric barrier discharge plasma hand sanitizer using air and tap water enriched by hydrogen peroxide," *Plasma Medicine*, vol. 7, no. 3, 2017.

[40] I. Osman, A. Ponukumati, M. Vargas, D. Bhakta, B. Ozoglu, and C. Bailey, "Plasma-activated vapor for sanitization of hands," *Plasma Medicine*, vol. 6, no. 3-4, 2016.

[41] Z. Chen, G. Garcia, V. Arumugaswami, and R. E. Wirz, "Cold atmospheric plasma for sars-cov-2 inactivation," *Physics of Fluids*, vol. 32, no. 11, 2020.

[42] J. J. Cotter, P. Maguire, F. Soberon, S. Daniels, J. P. O'Gara, and E. Casey, "Disinfection of meticillin-resistant staphylococcus aureus and staphylococcus epidermidis biofilms using a remote non-thermal gas plasma," *Journal of Hospital Infection*, vol. 78, no. 3, pp. 204–207, 2011.

[43] A. Filipić, I. Gutierrez-Aguirre, G. Primc, M. Mozetič, and D. Dobnik, "Cold plasma, a new hope in the field of virus inactivation," *Trends in Biotechnology*, vol. 38, no. 11, pp. 1278–1291, 2020.

[44] R. Hervé, M. G. Kong, S. Bhatt, H.-L. Chen, E. Comoy, J. Deslys, T. Secker, and C. Keevil, "Evaluation of cold atmospheric plasma for the decontamination of flexible endoscopes," *Journal of Hospital Infection*, vol. 136, pp. 100–109, 2023.

[45] M. H. Lee, B. J. Park, S. C. Jin, D. Kim, I. Han, J. Kim, S. O. Hyun, K.-H. Chung, and J.-C. Park, "Removal and sterilization of biofilms and planktonic bacteria by microwave-induced argon plasma at atmospheric pressure," *New Journal of Physics*, vol. 11, no. 11, p. 115022, 2009.

[46] G. Isbary, W. Stolz, T. Shimizu, R. Monetti, W. Bunk, H.-U. Schmidt, G. E. Morfill, T. Klämpfl, B. Steffes, H. Thomas, *et al.*, "Cold atmospheric argon plasma treatment may accelerate wound healing in chronic wounds: Results of an open retrospective randomized controlled study in vivo," *Clinical Plasma Medicine*, vol. 1, no. 2, pp. 25–30, 2013.

[47] S. Hartwig, S. Preissner, J. O. Voss, M. Hertel, C. Doll, R. Waluga, and J. D. Raguse, "The feasibility of cold atmospheric plasma in the treatment of complicated wounds in cranio-maxillo-facial surgery," *Journal of Cranio-Maxillofacial Surgery*, vol. 45, no. 10, pp. 1724–1730, 2017.

[48] S. Hartwig, C. Doll, J. O. Voss, M. Hertel, S. Preissner, and J. D. Raguse, "Treatment of wound healing disorders of radial forearm free flap donor sites using cold atmospheric plasma: a proof of concept," *Journal of Oral and Maxillofacial Surgery*, vol. 75, no. 2, pp. 429–435, 2017.

[49] M. Klebes, C. Ulrich, F. Kluschke, A. Patzelt, S. Vandersee, H. Richter, A. Bob, J. von Hutten, J. T. Krediet, A. Kramer, *et al.*, "Combined antibacterial effects of tissue-tolerable plasma and a modern conventional liquid antiseptic on chronic wound treatment," *Journal of Biophotonics*, vol. 8, no. 5, pp. 382–391, 2015.

[50] S. Bekeschus, K. Rödder, A. Schmidt, M. B. Stope, T. von Woedtke, V. Miller, A. Fridman, K.-D. Weltmann, K. Masur, H.-R. Metelmann, *et al.*, "Cold physical plasma selects for specific t helper cell subsets with distinct cells surface markers in a caspase-dependent and nf-κb-independent manner," *Plasma Processes and Polymers*, vol. 13, no. 12, pp. 1144–1150, 2016.

[51] S. R. Murthy, X. Cheng, T. Zhuang, L. Ly, O. Jones, G. Basadonna, M. Keidar, and J. Canady, "Bcl2a1 regulates canady helios cold plasma-induced cell death in triple-negative breast cancer," *Scientific Reports*, vol. 12, no. 1, p. 4038, 2022.

[52] S. Bekeschus, A. Mueller, V. Miller, U. Gaipl, and K.-D. Weltmann, "Physical plasma elicits immunogenic cancer cell death and mitochondrial singlet oxygen," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 2, no. 2, pp. 138–146, 2017.

[53] A. Lin, B. Truong, A. Pappas, L. Kirifides, A. Oubarri, S. Chen, S. Lin, D. Dobrynin, G. Fridman, A. Fridman, *et al.*, "Uniform nanosecond pulsed dielectric barrier discharge plasma enhances anti-tumor effects by induction of immunogenic cell death in tumors and stimulation of macrophages," *Plasma Processes and Polymers*, vol. 12, no. 12, pp. 1392–1399, 2015.

[54] J. Canady, S. Gordon, T. Zhuang, S. Wigh, W. Rowe, A. Shashurin, D. Chiu, S. Jones, K. Wiley, E. Cohen, *et al.*, "Cold atmospheric plasma (CAP) combined with chemo-radiation and cytoreductive surgery: the first clinical experience for stage IV metastatic

colon cancer," *Comprehensive Clinical Plasma Medicine: Cold Physical Plasma for Medical Application*, pp. 163–183, 2018.

[55] L. Ly, X. Cheng, S. R. Murthy, T. Zhuang, O. Z. Jones, G. Basadonna, M. Keidar, and J. Canady, "Canady cold plasma conversion system treatment: An effective inhibitor of cell viability in breast cancer molecular subtypes," *Clinical Plasma Medicine*, vol. 19, p. 100109, 2020.

[56] M. Laroussi, "Effects of low temperature plasmas on proteins," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 2, no. 3, pp. 229–234, 2018.

[57] F. Nejat, N.-S. Nabavi, M.-A. Nejat, H. Aghamollaei, and K. Jadidi, "Safety evaluation of the plasma on ocular surface tissue: an animal study and histopathological findings," *Clinical Plasma Medicine*, vol. 14, p. 100084, 2019.

[58] M. Dünnbier, A. Schmidt-Bleker, J. Winter, M. Wolfram, R. Hippler, K. Weltmann, and S. Reuter, "Ambient air particle transport into the effluent of a cold atmospheric-pressure argon plasma jet investigated by molecular beam mass spectrometry," *Journal of Physics D: Applied Physics*, vol. 46, no. 43, p. 435203, 2013.

[59] C. Chen, D. Liu, A. Yang, H.-L. Chen, and M. G. Kong, "Aqueous reactive oxygen species induced by He+o2 plasmas: Chemistry pathways and dosage control approaches," *Plasma Chemistry and Plasma Processing*, vol. 38, pp. 89–105, 2018.

[60] J. Shin and L. L. Raja, "Run-to-run variations, asymmetric pulses, and long time-scale transient phenomena in dielectric-barrier atmospheric pressure glow discharges," *Journal of Physics D: Applied Physics*, vol. 40, no. 10, p. 3145, 2007.

[61] D. Gidon, D. B. Graves, and A. Mesbah, "Predictive control of 2D spatial thermal dose delivery in atmospheric pressure plasma jets," *Plasma Sources Science and Technology*, vol. 28, no. 8, p. 085001, 2019.

[62] D. Gidon, D. B. Graves, and A. Mesbah, "Spatial thermal dose delivery in atmospheric pressure plasma jets," *Plasma Sources Science and Technology*, vol. 28, no. 2, p. 025006, 2019.

[63] D. Gidon, B. Curtis, J. A. Paulson, D. B. Graves, and A. Mesbah, "Model-based feedback control of a kHz-excited atmospheric pressure plasma jet," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 2, no. 2, pp. 129–137, 2017.

[64] A. D. Bonzanini, J. A. Paulson, and A. Mesbah, "Safe learning-based model predictive control under state-and input-dependent uncertainty using scenario trees," in *Proceedings of the 59th IEEE Conference on Decision and Control*, pp. 2448–2454, 2020.

[65] L. Lin and M. Keidar, "Machine learning controlled self-adaptive plasma medicine," in *Proceedings of the IEEE International Conference on Plasma Science*, pp. 561–561, 2020.

[66] A. Sharma and L. Palaniappan, "Improving diversity in medical research," *Nature Reviews Disease Primers*, vol. 7, no. 1, p. 74, 2021.

[67] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivative-free optimization*. SIAM, 2009.

[68] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *Journal of Global Optimization*, vol. 56, pp. 1247–1293, 2013.

[69] J. A. Paulson, F. Sorourifar, and A. Mesbah, "A tutorial on derivative-free policy learning methods for interpretable controller representations," in *Proceedings of the American Control Conference*, pp. 1295–1306, 2023.

[70] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[71] K. Shao, D. Romeres, A. Chakrabarty, and A. Mesbah, "Preference-guided Bayesian optimization for control policy learning: Application to personalized plasma medicine," in *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*, 2023.

[72] K. J. Chan, J. A. Paulson, and A. Mesbah, "Deep learning-based approximate nonlinear model predictive control with offset-free tracking for embedded applications," in *Proceedings of the American Control Conference*, pp. 3475–3481, 2021.

[73] D. Rodrigues, K. J. Chan, and A. Mesbah, "Data-driven adaptive optimal control under model uncertainty: an application to cold atmospheric plasmas," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 1, pp. 55–69, 2022.

[74] Y. Bao, K. J. Chan, A. Mesbah, and J. M. Velni, "Learning-based adaptive-scenario-tree model predictive control with probabilistic safety guarantees using Bayesian neural networks," in *Proceedings of the American Control Conference*, pp. 3260–3265, 2022.

[75] Y. Bao, K. J. Chan, A. Mesbah, and J. M. Velni, "Learning-based adaptive-scenario-tree model predictive control with improved probabilistic safety using robust Bayesian neural networks," *International Journal of Robust Nonlinear Control*, vol. 33, no. 5, pp. 3312–3333, 2023.

[76] K. J. Chan, G. Makrygiorgos, and A. Mesbah, "Towards personalized plasma medicine via data-efficient adaptation of fast deep learning-based MPC policies," in *Proceedings of the American Control Conference*, pp. 2769–2775, 2023.

[77] K. J. Chan, J. A. Paulson, and A. Mesbah, "Safe explorative Bayesian optimization – Towards personalized treatments in plasma medicine," in *Proceedings of the 62nd Conference on Decision and Control*, pp. 4106–4111, 2023.

[78] K. J. Chan, J. A. Paulson, and A. Mesbah, "A practical multi-objective learning framework for optimal hardware-software co-design of control-on-a-chip systems," *IEEE Transactions in Control Systems Technology*, 2024. *Under Review.*

[79] K. J. Chan, A. Stancampiano, K. N. Skinner, E. Robert, and A. Mesbah, "A cold atmospheric plasma sensor for identification and differentiation of biological tissues," *Scientific Reports*, 2024. *Submitted.*

[80] X. Lu, M. Laroussi, and V. Puech, "On atmospheric-pressure non-equilibrium plasma jets and plasma bullets," *Plasma Sources Science and Technology*, vol. 21, no. 3, p. 034005, 2012.

[81] A. Schutze, J. Y. Jeong, S. E. Babayan, J. Park, G. S. Selwyn, and R. F. Hicks, "The atmospheric-pressure plasma jet: a review and comparison to other plasma sources," *IEEE transactions on plasma science*, vol. 26, no. 6, pp. 1685–1694, 1998.

[82] X. Lu, Z. Jiang, Q. Xiong, Z. Tang, X. Hu, and Y. Pan, "An 11cm long atmospheric pressure cold plasma plume for applications of plasma medicine," *Applied Physics Letters*, vol. 92, no. 8, 2008.

[83] A. Yang, X. Wang, M. Rong, D. Liu, F. Iza, and M. G. Kong, "1-D fluid model of atmospheric-pressure rf He+O2 cold plasmas: Parametric study and critical evaluation," *Physics of Plasmas*, vol. 18, no. 11, 2011.

[84] C. Chen, D. Liu, Z. Liu, A. Yang, H. Chen, G. Shama, and M. Kong, "A model of plasma-biofilm and plasma-tissue interactions at ambient pressure," *Plasma Chemistry and Plasma Processing*, vol. 34, pp. 403–441, 2014.

[85] A. Mesbah and D. B. Graves, "Machine learning for modeling, diagnostics, and control of non-equilibrium plasmas," *Journal of Physics D: Applied Physics*, vol. 52, no. 30, p. 30LT02, 2019.

[86] P. Van Overschee and B. De Moor, *Subspace identification for linear systems: Theory – Implementation – Applications.* Springer Science & Business Media, 2012.

[87] S. A. Sapareto and W. C. Dewey, "Thermal dose determination in cancer therapy," *International Journal of Radiation Oncology Biology Physics*, vol. 10, no. 6, pp. 787–800, 1984.

[88] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pp. 507–523, Springer, 2011.

[89] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, “Scalable bayesian optimization using deep neural networks,” in *International Conference on Machine Learning*, pp. 2171–2180, PMLR, 2015.

[90] C. E. Rasmussen, C. K. Williams, *et al.*, *Gaussian processes for machine learning*, vol. 1. Springer, 2006.

[91] J. G. March, “Exploration and exploitation in organizational learning,” *Organization science*, vol. 2, no. 1, pp. 71–87, 1991.

[92] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, “Autoware on board: Enabling autonomous vehicles with embedded systems,” in *Proceedings of the ACM/IEEE 9th International Conference on Cyber-Physical Systems*, pp. 287–296, 2018.

[93] D. Mazzei, G. Montelisciani, G. Baldi, and G. Fantoni, “Changing the programming paradigm for the embedded in the IoT domain,” in *Proceedings of the IEEE 2nd World Forum Internet of Things*, pp. 239–244, 2015.

[94] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, “FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge,” in *Proceedings of the 56th Design Automation Conference*, pp. 1–6, 2019.

[95] A. Moradkhani, A. Broumandnia, and S. J. Mirabedini, “A portable medical device for detecting diseases using probabilistic neural network,” *Biomedical Signal Processing and Control*, vol. 71, p. 103142, 2022.

[96] M. Farahi, A. Casals, O. Sarrafzadeh, Y. Zamani, H. Ahmadi, N. Behbood, and H. Habibian, “Beat-to-beat fetal heart rate analysis using portable medical device and wavelet transformation technique,” *Heliyon*, vol. 8, no. 12, 2022.

[97] G. F. Franklin, J. D. Powell, M. L. Workman, *et al.*, *Digital control of dynamic systems*, vol. 3. Addison-wesley Menlo Park, CA, 1998.

[98] B. Houska, H. J. Ferreau, and M. Diehl, “ACADO toolkit – an open-source framework for automatic control and dynamic optimization,” *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[99] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)," *Optimization and Engineering*, vol. 20, pp. 769–809, 2019.

[100] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *International Journal of Control*, vol. 93, no. 1, pp. 13–29, 2020.

[101] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 685–704, 2015.

[102] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.

[103] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2011.

[104] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proceedings of the IEEE*, vol. 82, no. 7, pp. 967–989, 1994.

[105] G. De Michell and R. K. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349–365, 1997.

[106] Y. Nishikawa, N. Sannomiya, T. Ohta, and H. Tanaka, "A method for auto-tuning of PID control parameters," *Automatica*, vol. 20, no. 3, pp. 321–332, 1984.

[107] C. Hang, K. Astrom, and Q. Wang, "Relay feedback auto-tuning of process controllers – a tutorial review," *Journal of Process Control*, vol. 12, no. 1, pp. 143–162, 2002.

[108] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[109] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[110] M. Zanon and S. Gros, "Safe reinforcement learning using robust MPC," *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3638–3652, 2020.

[111] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[112] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, "Data-efficient autotuning with Bayesian optimization: An industrial control study," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 730–740, 2019.

[113] C. König, M. Turchetta, J. Lygeros, A. Rupenyan, and A. Krause, "Safe and efficient model-free adaptive control via Bayesian optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 9782–9788, 2021.

[114] M. Zhu, A. Bemporad, and D. Piga, "Preference-based MPC calibration," in *Proceedings of the European Control Conference*, pp. 638–645, 2021.

[115] M. Forgione, D. Piga, and A. Bemporad, "Efficient calibration of embedded MPC," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5189–5194, 2020.

[116] M. Zhu, D. Piga, and A. Bemporad, "C-GLISp: Preference-based global optimization under unknown constraints with applications to controller calibration," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 2176–2187, 2021.

[117] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995.

[118] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari, "Approximating explicit model predictive control using constrained neural networks," in *Proceedings of the American Control Conference*, pp. 1520–1527, 2018.

[119] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie, and M. Klaučo, "Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems," *Journal of Process Control*, vol. 116, pp. 80–92, 2022.

[120] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams, "Predictive entropy search for multi-objective Bayesian optimization," in *Proceedings of the International Conference on Machine Learning*, pp. 1492–1501, PMLR, 2016.

[121] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, pp. 369–395, 2004.

[122] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms – a comparative case study," in *Proceedings of the International Conference on Parallel Problem Solving from Nature*, pp. 292–301, 1998.

[123] E. Bakshy, L. Dworkin, B. Karrer, K. Kashin, B. Letham, A. Murthy, and S. Singh, "AE: A domain-agnostic platform for adaptive experimentation," in *Proceedings of the Conference on Neural Information Processing Systems*, pp. 1–8, 2018.

[124] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[125] S.-i. Amari, “Backpropagation and stochastic gradient descent method,” *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[126] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, “Predicting the computational cost of deep learning models,” in *Proceedings of the IEEE International Conference on Big Data*, pp. 3873–3882, 2018.

[127] A. Suardi, E. C. Kerrigan, and G. A. Constantinides, “Fast FPGA prototyping toolbox for embedded optimization,” in *Proceedings of the European Control Conference*, pp. 2589–2594, 2015.

[128] S. Lucia, D. Navarro, O. Lucia, P. Zometa, and R. Findeisen, “Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, 2017.

[129] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[130] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, “The sample average approximation method for stochastic discrete optimization,” *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.

[131] A. Smola and P. Bartlett, “Sparse greedy gaussian process regression,” *Advances in Neural Information Processing Systems*, vol. 13, 2000.

[132] J. Quinonero-Candela and C. E. Rasmussen, “A unifying view of sparse approximate Gaussian process regression,” *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[133] B. Lei, T. Q. Kirk, A. Bhattacharya, D. Pati, X. Qian, R. Arroyave, and B. K. Mallick, “Bayesian optimization with adaptive surrogate models for automated experimental design,” *npj Computational Materials*, vol. 7, no. 1, p. 194, 2021.

[134] R. Moriconi, M. P. Deisenroth, and K. Sesh Kumar, “High-dimensional bayesian optimization using low-dimensional feature spaces,” *Machine Learning*, vol. 109, pp. 1925–1943, 2020.

[135] S. Daulton, M. Balandat, and E. Bakshy, “Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 2187–2200, 2021.

[136] G. Makrygiorgos, A. D. Bonzanini, V. Miller, and A. Mesbah, “Performance-oriented model learning for control via multi-objective Bayesian optimization,” *Computers & Chemical Engineering*, vol. 162, p. 107770, 2022.

[137] D. da Silva, “Proprietades geraes,” *Journal de l’Ecole Polytechnique*, vol. 30, 1854.

[138] Y. Wang and S. Boyd, “Fast model predictive control using online optimization,” *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2009.

[139] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi: A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[140] A. Alessio and A. Bemporad, “A survey on explicit model predictive control,” in *Nonlinear Model Predictive Control* (L. Magni, D. M. Raimondo, and F. Allgöwer, eds.), pp. 345–369, Springer Berlin Heidelberg, 2009.

[141] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, “Multi-Parametric Toolbox 3.0,” in *Proceedings of the European Control Conference*, pp. 502–510, 2013. `http://control.ee.ethz.ch/~mpt`.

[142] P. K. Chu, J. Chen, L. Wang, and N. Huang, “Plasma-surface modification of biomaterials,” *Materials Science and Engineering R: Reports*, vol. 36, no. 5-6, pp. 143–206, 2002.

[143] D. Petlin, S. Tverdokhlebov, and Y. Anissimov, “Plasma treatment as an efficient tool for controlled drug release from polymeric materials: A review,” *Journal of Controlled Release*, vol. 266, pp. 57–74, 2017.

[144] Y. Gorbanev, C. Verlackt, S. Tinck, E. Tuenter, K. Foubert, P. Cos, and A. Bogaerts, “Combining experimental and modelling approaches to study the sources of reactive species induced in water by the COST RF plasma jet,” *Physical Chemistry Chemical Physics*, vol. 20, no. 4, pp. 2797–2808, 2018.

[145] L. Lin, D. Yan, T. Lee, and M. Keidar, “Self-adaptive plasma chemistry and intelligent plasma medicine,” *Advanced Intelligent Systems*, vol. 4, no. 3, p. 2100112, 2022.

[146] D. Bernardini and A. Bemporad, “Scenario-based model predictive control of stochastic constrained linear systems,” in *Proceedings of the 48th IEEE Conference on Decision and Control/28th Chinese Control Conference*, pp. 6333–6338, 2009.

[147] S. Lucia, T. Finkler, and S. Engell, “Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty,” *Journal of Process Control*, vol. 23, no. 9, pp. 1306–1319, 2013.

[148] E. Klintberg, J. Dahl, J. Fredriksson, and S. Gros, “An improved dual Newton strategy for scenario-tree MPC,” in *Proceedings of the IEEE 55th Conference on Decision and Control*, pp. 3675–3681, 2016.

[149] I. M. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 784–802, 1967.

[150] M. Laroussi, S. Bekeschus, M. Keidar, A. Bogaerts, A. Fridman, X. Lu, K. Ostrikov, M. Hori, K. Stapelmann, V. Miller, *et al.*, "Low-temperature plasma for biology, hygiene, and medicine: Perspective and roadmap," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 6, no. 2, pp. 127–157, 2022.

[151] Y. Lyu, L. Lin, E. Gjika, T. Lee, and M. Keidar, "Mathematical modeling and control for cancer treatment with cold atmospheric plasma jet," *Journal of Physics D: Applied Physics*, vol. 52, no. 18, p. 185202, 2019.

[152] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the International Conference on Machine Learning*, pp. 387–395, PMLR, 2014.

[153] D. Piga, M. Forgione, S. Formentin, and A. Bemporad, "Performance-oriented model learning for data-driven MPC design," *IEEE Control Systems Letters*, vol. 3, no. 3, pp. 577–582, 2019.

[154] F. Sorourifar, G. Makrygirgos, A. Mesbah, and J. A. Paulson, "A data-driven automatic tuning method for MPC under uncertainty using constrained Bayesian optimization," *IFAC-PapersOnLine*, vol. 54, no. 3, pp. 243–250, 2021.

[155] A. Marco, F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal, and S. Trimpe, "Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1557–1563, 2017.

[156] M. Turchetta, A. Krause, and S. Trimpe, "Robust model-free reinforcement learning with multi-objective Bayesian optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 10702–10708, 2020.

[157] A. D. Bonzanini, J. A. Paulson, D. B. Graves, and A. Mesbah, "Toward safe dose delivery in plasma medicine using projected neural network-based fast approximate NMPC," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5279–5285, 2020.

[158] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and O. Lucia, "Deep learning-based model predictive control for resonant power converters," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 1, pp. 409–420, 2020.

[159] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[160] E. Borgonovo and E. Plischke, "Sensitivity analysis: A review of recent advances," *European Journal of Operational Research*, vol. 248, no. 3, pp. 869–887, 2016.

[161] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[162] S. Marelli and B. Sudret, *UQLab: A framework for uncertainty quantification in Matlab*, pp. 2554–2563. ASCE, 2014.

[163] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, "BoTorch: A framework for efficient Monte-Carlo Bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21524–21538, 2020.

[164] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[165] D. Krishnamoorthy and F. J. Doyle, "Safe Bayesian optimization using interior-point methods—applied to personalized insulin dose guidance," *IEEE Control Systems Letters*, vol. 6, pp. 2834–2839, 2022.

[166] T. Zhang, Q. Li, C.-s. Zhang, H.-w. Liang, P. Li, T.-m. Wang, S. Li, Y.-l. Zhu, and C. Wu, "Current trends in the development of intelligent unmanned autonomous systems," *Frontiers of Information Technology & Electronic Engineering*, vol. 18, pp. 68–85, 2017.

[167] F. Berkenkamp, A. Krause, and A. P. Schoellig, "Bayesian optimization with safety constraints: Safe and automatic parameter tuning in robotics," *Machine Learning*, pp. 1–35, 2021.

[168] D. Krishnamoorthy and F. J. Doyle III, "Model-free real-time optimization of process systems using safe Bayesian optimization," *AIChE Journal*, vol. 69, no. 4, p. e17993, 2023.

[169] J.-M. Pouvesle and E. Robert, "Multimodal action of atmospheric pressure plasma jets for biological applications," in *ISPB 2017*, 2017.

[170] S. R. Chowdhury and A. Gopalan, "On kernelized multi-armed bandits," in *Proceedings of the 34th International Conference on on Machine Learning*, pp. 844–853, PMLR, 2017.

[171] J. A. Paulson and C. Lu, "COBALT: COnstrained Bayesian optimizAtion of computationaLly expensive grey-box models exploiting derivaTive information," *Computers & Chemical Engineering*, vol. 160, p. 107700, 2022.

[172] C. Lu and J. A. Paulson, “No-regret Bayesian optimization with unknown equality and inequality constraints using exact penalty functions,” *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 895–902, 2022.

[173] P. Van Overschee and B. De Moor, *Subspace identification for linear systems: Theory—Implementation—Applications*. Springer Science & Business Media, 2012.

[174] D. Spether, M. Scharpf, J. Hennenlotter, C. Schwentner, A. Neugebauer, D. Nüßle, K. Fischer, H. Zappe, A. Stenzl, F. Fend, *et al.*, “Real-time tissue differentiation based on optical emission spectroscopy for guided electrosurgical tumor resection,” *Biomedical Optics Express*, vol. 6, no. 4, pp. 1419–1428, 2015.

[175] J. Dinnes, J. Bamber, N. Chuchu, S. E. Bayliss, Y. Takwoingi, C. Davenport, K. Godfrey, C. O’Sullivan, R. N. Matin, J. J. Deeks, *et al.*, “High-frequency ultrasound for diagnosing skin cancer in adults,” *Cochrane Database of Systematic Reviews*, vol. 2018, no. 12, 1996.

[176] Y. Winetraub, A. V. Vleck, E. Yuan, I. Terem, J. Zhao, C. Yu, W. Chan, H. Do, S. Shevidi, M. Mao, J. Yu, M. Hong, E. Blankenberg, K. E. Rieger, S. Chu, S. Aasi, K. Y. Sarin, and A. de la Zerda, “Noninvasive virtual biopsy using micro-registered optical coherence tomography (OCT) in human subjects,” *Science Advances*, vol. 10, no. 15, p. eadi5794, 2024.

[177] R. F. Thompson, G. Valdes, C. D. Fuller, C. M. Carpenter, O. Morin, S. Aneja, W. D. Lindsay, H. J. Aerts, B. Agrimson, C. Deville Jr, *et al.*, “Artificial intelligence in radiation oncology: a specialty-wide disruptive transformation?,” *Radiotherapy and Oncology*, vol. 129, no. 3, pp. 421–426, 2018.

[178] M. L. Giger, “Machine learning in medical imaging,” *Journal of the American College of Radiology*, vol. 15, no. 3, pp. 512–520, 2018.

[179] A. Takiddin, J. Schneider, Y. Yang, A. Abd-Alrazaq, and M. Househ, “Artificial intelligence for skin cancer detection: scoping review,” *Journal of Medical Internet Research*, vol. 23, no. 11, p. e22934, 2021.

[180] K. Das, C. J. Cockerell, A. Patil, P. Pietkiewicz, M. Giulini, S. Grabbe, and M. Goldust, “Machine learning and its application in skin cancer,” *International Journal of Environmental Research and Public Health*, vol. 18, no. 24, p. 13409, 2021.

[181] Y. A. Glickman, O. Filo, M. David, A. Yayon, M. Topaz, B. Zamir, A. Ginzburg, D. Rozenman, and G. Kenan, “Electrical impedance scanning: a new approach to skin cancer diagnosis,” *Skin Research and Technology*, vol. 9, no. 3, pp. 262–268, 2003.

[182] I. Bürger, M. Scharpf, J. Hennenlotter, D. Nüßle, D. Spether, A. Neugebauer, N. Bibinov, A. Stenzl, F. Fend, M. Enderle, *et al.*, “Tissue differentiation by means of high

resolution optical emission spectroscopy during electrosurgical intervention," *Journal of Physics D: Applied Physics*, vol. 50, no. 3, p. 035401, 2016.

[183] B. K. Park, Y. Woo, D. Jeong, J. Park, T.-Y. Choi, D. P. Simmons, J. Ha, and D. Kim, "Thermal conductivity of biological cells at cellular level and correlation with disease state," *Journal of Applied Physics*, vol. 119, no. 22, 2016.

[184] M. A. Özdemir, G. D. Özdemir, M. Gül, O. Güren, and U. K. Ercan, "Machine learning to predict the antimicrobial activity of cold atmospheric plasma-activated liquids," *Machine Learning: Science and Technology*, vol. 4, no. 1, p. 015030, 2023.

[185] Y. Yue, X. Pei, D. Gidon, F. Wu, S. Wu, and X. Lu, "Investigation of plasma dynamics and spatially varying O and OH concentrations in atmospheric pressure plasma jets impinging on glass, water and metal substrates," *Plasma Sources Science and Technology*, vol. 27, no. 6, p. 064001, 2018.

[186] C.-Y. Wang and C.-C. Hsu, "Development and testing of an efficient data acquisition platform for machine learning of optical emission spectroscopy of plasmas in aqueous solution," *Plasma Sources Science and Technology*, vol. 28, no. 10, p. 105013, 2019.

[187] T. Teschner, R. Bansemer, K.-D. Weltmann, and T. Gerling, "Investigation of power transmission of a helium plasma jet to different dielectric targets considering operating modes," *Plasma*, vol. 2, no. 3, pp. 348–359, 2019.

[188] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[189] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[190] L. Martinez, A. Dhruv, L. Lin, E. Balaras, and M. Keidar, "Interaction between a helium atmospheric plasma jet and targets and dynamics of the interface," *Plasma Sources Science and Technology*, vol. 28, no. 11, p. 115002, 2019.

[191] S. Gabriel, R. Lau, and C. Gabriel, "The dielectric properties of biological tissues: II. measurements in the frequency range 10 hz to 20 ghz," *Physics in Medicine & Biology*, vol. 41, no. 11, p. 2251, 1996.

[192] K. Cho, K. Shao, and A. Mesbah, “Run-indexed time-varying Bayesian optimization with positional encoding for auto-tuning of controllers: Application to a plasma-assisted deposition process with run-to-run drifts,” *Computers & Chemical Engineering*, vol. 185, p. 108653, 2024.

[193] T. T. Nguyen and V. J. Reddi, “Deep reinforcement learning for cyber security,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 8, pp. 3779–3795, 2021.

[194] J. Nayak, S. K. Meher, A. Souri, B. Naik, and S. Vimal, “Extreme learning machine and bayesian optimization-driven intelligent framework for iomt cyber-attack detection,” *The Journal of Supercomputing*, vol. 78, no. 13, pp. 14866–14891, 2022.

[195] M. Injadat, F. Salo, A. B. Nassif, A. Essex, and A. Shami, “Bayesian optimization with machine learning algorithms towards anomaly detection,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.

[196] J. Trieschmann, L. Vialetto, and T. Gergs, “Machine learning for advancing low-temperature plasma modeling and simulation,” *Journal of Micro/Nanopatterning, Materials, and Metrology*, vol. 22, no. 4, pp. 041504–041504, 2023.

[197] A. D. Bonzanini, K. Shao, D. B. Graves, S. Hamaguchi, and A. Mesbah, “Foundations of machine learning for low-temperature plasmas: methods and case studies,” *Plasma Sources Science and Technology*, vol. 32, no. 2, p. 024003, 2023.

[198] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[199] H. S. Pillai, Y. Li, S.-H. Wang, N. Omidvar, Q. Mu, L. E. Achenie, F. Abild-Pedersen, J. Yang, G. Wu, and H. Xin, “Interpretable design of Ir-free trimetallic electrocatalysts for ammonia oxidation with graph neural networks,” *Nature Communications*, vol. 14, no. 1, p. 792, 2023.

[200] J. A. Esterhuizen, B. R. Goldsmith, and S. Linic, “Interpretable machine learning for knowledge generation in heterogeneous catalysis,” *Nature Catalysis*, vol. 5, no. 3, pp. 175–184, 2022.

[201] T. D. Holmes, R. H. Rothman, and W. B. Zimmerman, “Graph theory applied to plasma chemical reaction engineering,” *Plasma Chemistry and Plasma Processing*, vol. 41, pp. 531–557, 2021.

[202] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[203] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[204] I. D. J. Rodriguez, A. Ames, and Y. Yue, "Lyanet: A lyapunov framework for training neural odes," in *International Conference on Machine Learning*, pp. 18687–18703, PMLR, 2022.

[205] G. Isbary, G. Morfill, H. Schmidt, M. Georgi, K. Ramrath, J. Heinlin, S. Karrer, M. Landthaler, T. Shimizu, B. Steffes, *et al.*, "A first prospective randomized controlled trial to decrease bacterial load using cold atmospheric argon plasma on chronic wounds in patients," *British Journal of Dermatology*, vol. 163, no. 1, pp. 78–82, 2010.

[206] M. Laroussi, "Cold plasma in medicine and healthcare: The new frontier in low temperature plasma applications," *Frontiers in Physics*, vol. 8, p. 74, 2020.

[207] A. Takahashi and T. Suzuki, "Bayesian optimization design for finding a maximum tolerated dose combination in phase i clinical trials," *The International Journal of Biostatistics*, vol. 18, no. 1, pp. 39–56, 2022.

[208] J. Richter, T. Friede, and J. Rahnenführer, "Improving adaptive seamless designs through bayesian optimization," *Biometrical Journal*, vol. 64, no. 5, pp. 948–963, 2022.

[209] P. Wegner, "Concepts and paradigms of object-oriented programming," *ACM Sigplan Oops Messenger*, vol. 1, no. 1, pp. 7–87, 1990.

[210] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, J. D. Siirola, *et al.*, *Pyomo-optimization modeling in python*, vol. 67. Springer, 2017.

[211] F. Fiedler, B. Karg, L. Lüken, D. Brandner, M. Heinlein, F. Brabender, and S. Lucia, "do-mpc: Towards fair nonlinear and robust model predictive control," *Control Engineering Practice*, vol. 140, p. 105676, 2023.

[212] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. v. Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, no. 1, pp. 147–183, 2022.

Appendix A

Open-source Code Repositories

Table A.1 lists some of the open-source code repositories that have been created over the course of this thesis.

Description	**Repository Link**
Code for [78]: End-to-end hardware-software co-design for embedded control policies in plasma medicine	`https://github.com/Mesbah-Lab-UCB/HW-SW_CoDesign4CoC`
Code for [76]: Personalized medicine via adaptation of deep learning-based approximate MPC	`https://github.com/kchan45/BO4Policy_Search_Plasma`
Code for [77]: Safe exploration for adaptation of (robust) MPC policies	`https://github.com/kchan45/SafeBOPlasma`
Code for [79]: Identification and characterization of biological tissues using CAPs	`https://github.com/Mesbah-Lab-UCB/CAP-Sensor4Bio`
Code to facilitate data collection in [79]	`https://github.com/kchan45/PlasmaGun`
Code for [73]: Data-driven adaptation of control policies under model uncertainty	`https://github.com/dfmrodrigues/SNSF-project-P2ELP2_184521/tree/main/Optimal%20control`
MPC code tutorials with CasADi and some object-oriented aspects	`https://github.com/kchan45/MPC_tutorials`
A partial template and/or example of code for conducting real-time experiments	`https://github.com/kchan45/Plasma-Wafers`
Latest aggregation of codes and documentation for the CAPJ testbed	`https://github.com/kchan45/Mesbah-APPJ`
Past repositories of code and documentation for the CAPJ testbed	`https://github.com/dgngdn/APPJ_Control` and `https://github.com/adbonzanini/APPJ-MacOS-Communication`
Collection of codes used in Berkeley-Lam collaborations	`https://github.com/kchan45/Berkeley-Lam-2023-UNLOCK`
Basic code to implement deep neural networks on field programmable gate arrays	`https://github.com/kchan45/DNN_MPC_Plasma_FPGA`

Table A.1: Code repositories created and/or used in this thesis.

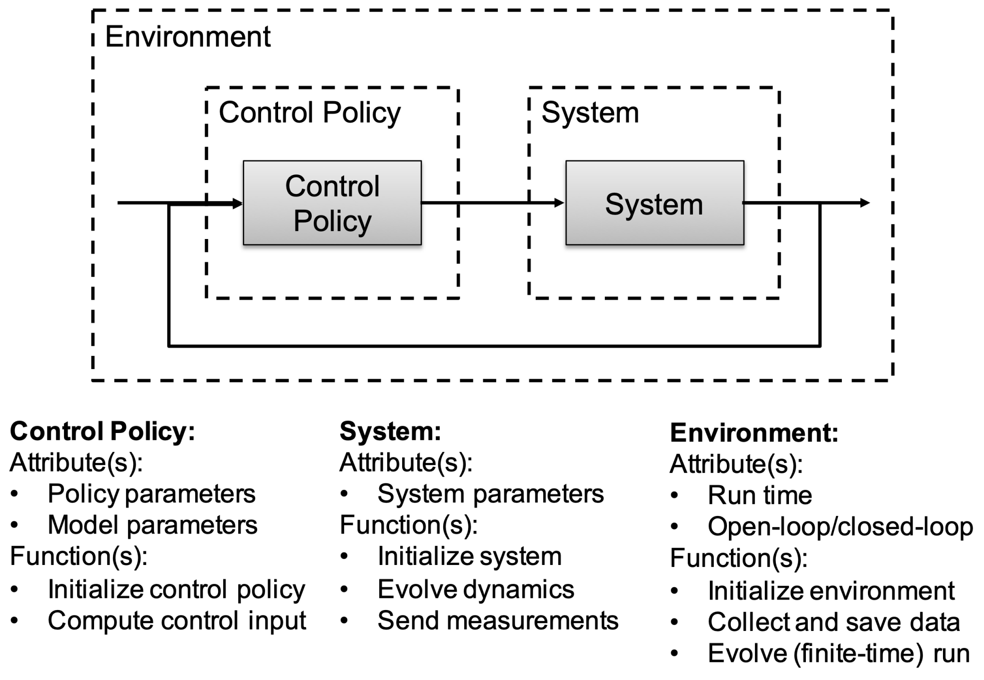
Appendix B

Software Design for Run-to-run Control Policy Adaptation

As mentioned in Chapter 3, the structure of the control in this dissertation relies on a hierarchical or multi-layered approach. Doing so requires software design that allows for interchangeable separation of each component of the system. This dissertation followed an object-oriented programming-inspired approach to developing the software to enable the third level of data-driven optimization. This appendix will describe the structure and implementation of such codes used throughout the dissertation.

B.1 Object-Oriented Programming for Closed-loop Control

Object oriented programming (OOP) is a programming paradigm that relies on the organization of code into "objects," or things that have specific attributes and functions [209]. One major advantage to the OOP structure involves modularity, allowing for objects to be swapped out without having to change or re-develop the whole system. This means that we define a standardized technique, which enables interchangeable blocks of code. In a feedback control loop, there are three main components that can be subdivided into objects: the control policy, the system, and the environment. Figure B.1 visually represents this subdivision. Each of the objects are broken down to have particular attributes and functions. For example, a function of the control policy should involve receiving feedback and computing the next input. Then, functions of the system may involve receiving input(s), evolving the dynamics, and outputting measurements. Finally, the function of the environment may be to connect the outputs of the system to the controller, aka "close the loop." Note that this is not a new concept for control systems, and there are a few recent Python packages that implement this coding paradigm for MPC-type policies, e.g., Pyomo [210], doMPC [211], acados [212]. However, these packages were limited during the initial years of this thesis,



Control Policy:
Attribute(s):
- Policy parameters
- Model parameters

Function(s):
- Initialize control policy
- Compute control input

System:
Attribute(s):
- System parameters

Function(s):
- Initialize system
- Evolve dynamics
- Send measurements

Environment:
Attribute(s):
- Run time
- Open-loop/closed-loop

Function(s):
- Initialize environment
- Collect and save data
- Evolve (finite-time) run

Figure B.1: An illustration of how a typical feedback control loop can be broken down into components for an object-oriented programming approach. The typical feedback control loop consists of the shaded boxes and arrows. The dashed line boxes indicate the segregation of those components as "objects." The list of attributes and functions are of each object type and are not exhaustive.

particularly in (i) robust/stochastic implementations of MPC and (ii) integration with experimental setups. This chapter aims to describe the methodology to give insight on inner workings of this dissertation.

B.2 A Python Implementation of MPC Policies using CasADi

CasADi is a widely used and recognized algorithmic differentiation and nonlinear optimization software [139]. This dissertation used CasADi as the backbone to develop MPC policies. The primary structure of an MPC-type policy is shown in the following Python code script:

```python
# This script provides definitions of classes that can be used for model
# predictive control (MPC) schemes.
#
#
```

```python
# Requirements:
# * Python 3
# * CasADi [https://web.casadi.org]
#
# Copyright (c) 2024 Mesbah Lab. All Rights Reserved.
# Contributor(s): Kimberly Chan
#
# This file is under the MIT License. A copy of this license is
    included in the
# download of the entire code package (within the root folder of the
    package).

import sys
sys.dont_write_bytecode = True
import casadi as cas
import numpy as np

GENERIC_P_OPTS = {
    "verbose": 0,
    "expand": True,
    "print_time": 0,
}  # problem options

GENERIC_S_OPTS = {
    "max_iter": 1000,
    "print_level": 0,
    "tol": 1e-6,
}  # solver options


class MPC:
    """
    MPC is a super class designed to be a template for particular
    implementations of model predictive controllers (MPCs). Users
   should develop
    their own MPCs by using the general structure/methods provided
   below. Upon
    initialization of this class or any of its child classes, users
   should
    provide a Python dict that contains all of the relevant problem
   information.
    This class is designed to be used with CasADi to generate the
   optimization
    problems associated with MPC. Users are referred to the CasADi
   documentation
    for more information on CasADi.
    """

    def __init__(self, prob_info):
        super(MPC, self).__init__()
        self.prob_info = prob_info
        self.mpc_type = None
        self.opti = None
        self.opti_vars = {}
        self.opti_params = {}

    def get_mpc(self, arg):
        """
        This method should generate the MPC problem by unpacking relevant
        information from the prob_info dict defined upon instantiation of the
        class. Upon defining the optimization problem, this method should save
        and/or return the appropriate objects such that this object may be
        called upon later. (e.g. if using the Opti stack interface of CasADi,
        save/return the objects that reference the optimization object
        (typically named opti) and the optimization variable references)
        """
        raise NotImplementedError

    def reset_initial_guesses(self, arg):
        """
        This method should reset any initial guesses of the decision variables
        passed into the optimization problem. This method provides a way to
        simulate repeated solves of the optimization problem without re-defining
        an entirely new problem. If using the Opti stack interface of CasADi,
        this method mainly involves using the set_initial() method of the Opti
        object.
        """
        raise NotImplementedError

    def set_parameters(self, arg):
        """
        This method should (re)set any parameters in the optimization problem.
        This method provides a way to simulated consistent and repeated solves
        of the optimization problem without re-defining an entirely new problem.
        If using the Opti stack interface of CasADi, this method mainly involves
        using the set_value() method of the Opti object.
        """
        raise NotImplementedError

    def solve_mpc(self, arg):
        """
        This method should solve the optimization problem and return/save the
        relavent optimal variables. For MPC, this is typically the first optimal
        input determined by the solver. Users may also wish to return other
        values of the optimization problem and/or the entire solution of the
        optimization problem. This method should also handle any Exceptions that
        may occur upon a call to solve the optimization problem in the form of a
        try/except clause. If using the Opti stack interface of CasADi, this
        method mainly involves the call to the solve() method of the Opti
        object, as well as calls to the value() method of OptiSolution/Opti
        objects.
        """
        pass

    def get_control_inputs(self, arg):
        """
        This method is a wrapper around the optimization problem to interface
        with simulation or experiments. It should 1) set the initial state and
        any other parameters (which may involve casting the true states to
        deviation variables) for the optimal control problem (OCP), 2) solve
        the OCP, 3) return the optimal input for the true system.
        """
        raise NotImplementedError

    def get_params(self):
        """
        This method will return the parameters of the optimal control problem (OCP) if the OCP has been defined
        """
        if self.opti_params:
            return self.opti_params.keys()
        else:
            raise ValueError("The OCP has not been created. There are no parameters to list!")
```

CasADi provides the `Opti` interface, which is a user-friendly take on defining optimization problems in code and follows a more natural syntax based on the mathematical formulation. The following script provides an example implementation of a non-robust nonlinear tracking problem that was often used in the plasma thermal dose control policy definition.

```
from controller import *

class NonlinearTrackingMPC(MPC):

    def __init__(self, prob_info):
        super(NonlinearTrackingMPC, self).__init__(prob_info)
        self.mpc_type = 'nonlinear-tracking'

    def get_mpc(self, p_opts=GENERIC_P_OPTS, s_opts=GENERIC_S_OPTS):
        """
        This method creates the optimization problem for the MPC. All
        information necessary for the creation of this controller is passed upon
        instantiation of this object within the prob_info dict. For more details
        on the optimization problem, the user is referred to the paper
        associated with the release of this code.

        This code uses IPOPT for the NLP solver which is distributed with
        CasADi. Users are referred to IPOPT [https://coin-or.github.io/Ipopt/]
        and the associated paper for more information on this solver.
        """
        # unpack relavant problem information
        Np = self.prob_info["Np"]

        nu = self.prob_info["nu"]
        nx = self.prob_info["nx"]
        ny = self.prob_info["ny"]

        u_min = self.prob_info["u_min"]
        u_max = self.prob_info["u_max"]
        x_min = self.prob_info["x_min"]
        x_max = self.prob_info["x_max"]
        y_min = self.prob_info["y_min"]
        y_max = self.prob_info["y_max"]

        u_init = self.prob_info["u_init"]
        x_init = self.prob_info["x_init"]
        y_init = self.prob_info["y_init"]

        f = self.prob_info["f"]
        h = self.prob_info["h"]
        lstage = self.prob_info["lstage"]
        lterm = self.prob_info["lterm"]
        reduce_dinput = False
        constrain_dinput = False
        if "ustage" in self.prob_info.keys():
            ustage = self.prob_info["ustage"]
            reduce_dinput = True
        if "du_max" in self.prob_info.keys():
            du_max = self.prob_info["du_max"]
            du_min = self.prob_info["du_min"]
            constrain_dinput = True

        # create NLP opti object
        opti = cas.Opti()

        # Initialize container lists for all states, inputs, outputs,
    and
        # predicted noise over horizon
        X = [0]*(Np+1)
        Ref = [0]*(Np+1)
        Y = [0]*(Np+1)
        U = [0]*Np

        J = 0  # initialize cost/objective function

        # define parameter(s), variable(s), and problem
        X[0] = opti.parameter(nx)  # initial state as a parameter
        opti.set_value(X[0], np.zeros((nx, 1)))

        # define reference parameter (for state ref, output ref, or
    single ref)
        if self.prob_info["state_ref"]:
            Ref[0] = opti.parameter(nx)
            opti.set_value(Ref[0], np.zeros((nx, 1)))
        elif self.prob_info["output_ref"]:
            Ref[0] = opti.parameter(ny)
            opti.set_value(Ref[0], np.zeros((ny, 1)))
        else:
            Ref[0] = opti.parameter(1)
            opti.set_value(Ref[0], 0.0)

        Y[0] = opti.variable(ny)  # initial output variable
        opti.subject_to(Y[0] == h(X[0]))
        opti.set_initial(Y[0], y_init)

        # the loop below systematically defines the variables of the
    optimal
        # control problem (OCP) over the prediction horizon
        for k in range(Np):
```

```python
            # variable and constraints for u_{k}
            U[k] = opti.variable(nu)
            opti.subject_to(opti.bounded(u_min, U[k], u_max))
            opti.set_initial(U[k], u_init)

            # define reference parameter (for state ref, output ref, or single ref)
            if self.prob_info["state_ref"]:
                Ref[k] = opti.parameter(nx)
                opti.set_value(Ref[k], np.zeros((nx, 1)))
            elif self.prob_info["output_ref"]:
                Ref[k] = opti.parameter(ny)
                opti.set_value(Ref[k], np.zeros((ny, 1)))
            else:
                Ref[k] = opti.parameter(1)
                opti.set_value(Ref[k], 0.0)

            # get stage cost
            Jstage = lstage(X[k], U[k], Ref)
            J += Jstage # add to the cost (construction of the objective)

            # variable x_{k+1}
            X[k + 1] = opti.variable(nx)
            opti.subject_to(opti.bounded(x_min, X[k + 1], x_max))
            opti.set_initial(X[k + 1], x_init)

            # variable y_{k+1}
            Y[k + 1] = opti.variable(ny)
            opti.subject_to(opti.bounded(y_min, Y[k + 1], y_max))
            opti.set_initial(Y[k + 1], y_init)

            # dynamics constraint
            opti.subject_to(X[k + 1] == f(X[k], U[k]))

            # output equality constraint
            opti.subject_to(Y[k + 1] == h(X[k + 1]))

            if k > 0 and reduce_dinput:
                J += ustage(U[k], U[k - 1])
            elif k > 0 and constrain_dinput:
                opti.subject_to(
                    opti.bounded(du_min, U[k] - U[k - 1], du_max)
                )

        # terminal cost and constraints
        J_end = lterm(X[-1], Ref[-1])
        J += J_end

        # set to minimize objective/cost
        opti.minimize(J)

        opti.solver("ipopt", p_opts, s_opts)  # add the solver to the opti object

        # save list containers of variables/parameters into a dict for portability
        opti_vars = {}
        opti_vars["U"] = U
        opti_vars["X"] = X[1:]
        opti_vars["Y"] = Y
        opti_vars["J"] = J

        opti_params = {}
        opti_params["X0"] = X[0]
        opti_params["Ref"] = Ref

        # save opti object and variable containers as attributes of object
        self.opti = opti
        self.opti_vars = opti_vars
        self.opti_params = opti_params

        return opti, opti_vars, opti_params

    def solve_mpc(self, warm_start=True):
        """
        This method solves the MPC as defined by the get_mpc() method of this
        class. This method can only be called after the the get_mpc() method has
        been called (i.e., the optimization problme must be defined before it
        can be solved).
        """
        # extract all keys from the opti variables dict
        opti_var_keys = [*self.opti_vars]
        opti_param_keys = [*self.opti_params]

        # unpack relevant information from problem creation
        u_min = self.prob_info['u_min']
        u_max = self.prob_info['u_max']

        feas = True
        res = {}
        try:
            sol = self.opti.solve()

            # process solution of optimization problem output
            for key in opti_var_keys:
                # if key is J, then the value is a single scalar
                if key == 'J':
                    res[key] = sol.value(self.opti_vars[key])
                # otherwise, a vector (or matrix) must be constructed
                else:
                    var = self.opti_vars[key]
                    r = len(var) # Np
                    nx = (var[0]).size1()
                    values = np.zeros((nx,r))
                    for j in range(r):
                        values[:,j] = sol.value(var[j])
                    res[key] = values

            res["Ufull"] = res["U"]  # the full optimal input trajectory is saved as Ufull
            res['U'] = res['U'][:,0] # the first optimal input is saved as U

            for key in opti_param_keys:
                # if key is X0, then the value is a single vector
                if key == 'X0':
                    res[key] = sol.value(self.opti_vars[key])
                # otherwise, a vector (or matrix) must be constructed
                else:
                    var = self.opti_params[key]
                    r = len(var) # Np
                    nx = (var[0]).size1()
                    values = np.zeros((nx,r))
                    for j in range(r):
                        values[:,j] = sol.value(var[j])
                    res[key] = values

            if warm_start:
                self.opti.set_initial(sol.value_variables())

        except Exception as e:
            print(e)
            # if solve fails, get the last value
            feas = False

            # process solution of optimization problem output
            for key in opti_var_keys:
                # if key is J, then the value is a single scalar
                if key == 'J':
                    res[key] = self.opti.debug.value(self.opti_vars[key])
                # otherwise, a vector (or matrix) must be constructed
                else:
                    var = self.opti_vars[key]
                    r = len(var) # Np
                    nx = (var[0]).size1()
                    values = np.zeros((nx,r))
                    for j in range(r):
                        values[:, j] = self.opti.debug.value(var[j])
                    res[key] = values

            res["Ufull"] = res["U"]  # the full optimal input trajectory is saved as Ufull
            res['U'] = res['U'][:,0] # the first optimal input is saved as U

            for key in opti_param_keys:
                # if key is X0, then the value is a single vector
                if key == 'X0':
                    res[key] = self.opti.debug.value(self.opti_vars[key])
                # otherwise, a vector (or matrix) must be constructed
                else:
                    var = self.opti_params[key]
                    r = len(var) # Np
                    nx = (var[0]).size1()
                    values = np.zeros((nx,r))
                    for j in range(r):
                        values[:, j] = self.opti.debug.value(var[j])
                    res[key] = values

            u = res['U']
            res['U'] = np.maximum(np.minimum(u, u_max), u_min)
            # print('U_0:', res['U'])
            # print('J:', res['J'])

        return res, feas

    def reset_initial_guesses(self):
        """
        This method resets the intial guesses for the variables of the
        optimization problem back to those defined in the problem_info dict
        provided upon instantiation of the OffsetFreeMPC object.
        """
        # unpack relevant information from the prob_info dict
        Np = self.prob_info["Np"]
        u_init = self.prob_info["u_init"]
        x_init = self.prob_info["x_init"]
        y_init = self.prob_info["y_init"]

        # unpack relevant variable containers from problem creation
        U = self.opti_vars["U"]
        X = self.opti_vars["X"]
        Y = self.opti_vars["Y"]
```

```python

        self.opti.set_initial(Y[0], y_init)
        for k in range(Np):
            self.opti.set_initial(U[k], u_init)
            self.opti.set_initial(X[k], x_init)
            self.opti.set_initial(Y[k + 1], y_init)

    def set_parameters(self, params_list):
        """
        This method sets the values of the parameters of the optimization
        problem to those provided as arguments to this method. The argument
        params_list is a list of new parameter values to set in the same order
        as the opti_param keys.
        """
        Np = self.prob_info["Np"]

        # unpack parameter containers
        X0 = self.opti_params["X0"]
        Ref = self.opti_params["Ref"]

        # unpack params_list argument
        x0 = params_list[0]
        ref = params_list[1]

        # reset initial condition
        self.opti.set_value(X0, x0)

        # set reference trajectory
        self.opti.set_value(Ref[0], ref[0])
        for k in range(Np):
            self.opti.set_value(Ref[k], ref[k])

    def get_control_inputs(self, params_list, transform_x0=True):
        if "yss" in self.prob_info.keys() and transform_x0:
            params_list[0] -= self.prob_info["yss"]

        self.set_parameters(params_list)
        res, feas = self.solve_mpc()

        uopt = res["U"] + self.prob_info["uss"]
        return uopt, res, feas
```

B.3 Transferring Implementations from Simulation to Experiment

A closed-loop simulation consists of each of the parts described in Section B.1, where the environment is the overall closed-loop simulation environment, aka an *in silico* experiment. The following code represents an example of the definition of the closed-loop simulation environment and was taken from the code used for [76]. Note that in this example, the "system" was not separated into its own object and the system parameters are built into the problem configuration rather than as its own object.

```python
# simulation functions
#
# This file defines a Simulation class to be used for closed loop simulations of
# model predictive controllers (MPCs) generated via the MPC subclasses or via
# deep neural network approximations of MPC controllers.
#
# Requirements:
# * Python 3
#
# Copyright (c) 2021 Mesbah Lab. All Rights Reserved.
# Kimberly Chan
#
# This file is under the MIT License. A copy of this license is included in the
# download of the entire code package (within the root folder of the package).

import sys
sys.dont_write_bytecode = True
import numpy as np
import time
from numpy.random import default_rng

import KCutils

class Simulation():
    """
    The Simulation class is used to create a simulation 'environment' defined by
    given problem information.
    """

    def __init__(self, Nsim):
        """
        Instantiation of the Simulation class requires the input argument Nsim,
        which denotes the length of the simulation horizon.
        """
        super(Simulation, self).__init__()
        self.Nsim = Nsim
        self.prob_info = None
        self.rand_seed = None

    def load_prob_info(self, prob_info):
        """
        This method loads the relevant problem information for
simulation and
        assigns them as attributes of the class from the prob_info dict
 used by
        other classes included in this package.
        """
        # extract and save relevant problem information
        self.prob_info = prob_info

        # system sizes
        self.nu = prob_info['nu']
        self.nx = prob_info['nx']
        self.ny = prob_info['ny']
        self.nyc = prob_info['nyc']
        self.nv = prob_info['nv']
        self.nw = prob_info['nw']
        self.nd = prob_info['nd']

        # disturbance/noise minimums and maximums
        if 'v_mu' in prob_info.keys() and 'v_sigma' in prob_info.keys()
:
            self.v_mu = prob_info['v_mu']
            self.v_sigma = prob_info['v_sigma']
            self.v_noise_generation = 'normal'
        elif 'v_max' in prob_info.keys() and 'v_min' in prob_info.keys
():
            self.v_min = prob_info['v_min']
            self.v_max = prob_info['v_max']
            self.v_noise_generation = 'uniform'
        else:
            print('No noise bounds/parameters given. Assuming no
measurement noise...')
            self.v_min = np.zeros((nv,))
            self.v_max = np.zeros((nv,))
            self.v_noise_generation = 'uniform'

        if 'w_mu' in prob_info.keys() and 'w_sigma' in prob_info.keys()
:
            self.v_mu = prob_info['w_mu']
            self.v_sigma = prob_info['w_sigma']
            self.v_noise_generation = 'normal'
```

```python
        elif 'w_max' in prob_info.keys() and 'w_min' in prob_info.keys():
            self.w_min = prob_info['w_min']
            self.w_max = prob_info['w_max']
            self.w_noise_generation = 'uniform'
        else:
            print('No noise bounds/parameters given. Assuming no process noise...')
            self.w_min = np.zeros((nw,))
            self.w_max = np.zeros((nw,))
            self.w_noise_generation = 'uniform'

        self.x0 = prob_info['x0'] # initial state/point
        self.hp = prob_info['hp'] # output equation for the 'real' system (plant)
        self.fp = prob_info['fp'] # dynamics equation for the plant
        self.myref = prob_info['myref'] # reference function for the controlled output
        self.ts = prob_info['ts'] # simulation sampling time
        self.rand_seed = prob_info['rand_seed'] # random seed

    def run_closed_loop(self, c,
                        observer=None,
                        offset=False,
                        CEM=False,
                        multistage=False,
                        rand_seed2=0):
        """
        This method runs a closed-loop simulation of the system using
        information derived from loading problem information and a controller
        (implicit MPC or approximate control). The problem information must be
        loaded before a closed-loop simulation can occur. The argument c is a
        controller object created using one of the classes defined in
        controller.py (for an MPC) or neural_network.py (for a DNN approximation
        to a MPC law).
        """
        # check to ensure problem data has been loaded
        if self.prob_info is None:
            print('Problem data not loaded. Please load the appropriate problem data by running the load_prob_info method.')
            raise

        # check controller type
        mpc_controller = False
        if isinstance(c, KCutils.controller.MPC):
            mpc_controller = True
            print('Using a MPC.')
        elif isinstance(c, KCutils.neural_network.DNN) or isinstance(c, KCutils.neural_network.SimpleDNN):
            print('Using an approximate controller.')
        else:
            print('Unsupported controller type.')
            raise

        if multistage:
            Wset = self.prob_info['Wset']

        # create a random number generator (RNG) to use for generating
        # noise/disturbances; use the same seed for consistent simulations
        if self.rand_seed is None:
            rng = default_rng()
        else:
            rng = default_rng(self.rand_seed+rand_seed2)

        if self.v_noise_generation == 'uniform':
            Vsim = rng.random(size=(self.nv,self.Nsim+1)) * (self.v_max-self.v_min)[:,None] + self.v_min[:,None]
        elif self.v_noise_generation == 'normal':
            Vsim = rng.normal(self.v_mu, self.v_sigma, size=(self.nv, self.Nsim+1))
        else:
            print('Unknown/unsupported form of noise generation!')
            raise
        if self.w_noise_generation == 'uniform':
            Wsim = rng.random(size=(self.nw,self.Nsim)) * (self.w_max-self.w_min)[:,None] + self.w_min[:,None]
        elif self.w_noise_generation == 'normal':
            Wsim = rng.normal(self.w_mu, self.w_sigma, size=(self.nw, self.Nsim+1))
        else:
            print('Unknown/unsupported form of noise generation!')
            raise

        # instantiate container variables for storing simulation data
        Xsim = np.zeros((self.nx,self.Nsim+1)) # state trajectories (plant)
        Ysim = np.zeros((self.ny,self.Nsim+1)) # output trajectories (plant)
        Usim = np.zeros((self.nu,self.Nsim))   # input trajectories (plant)
        Xhat = np.zeros_like(Xsim)  # state estimation
        Dhat = np.zeros((self.nd,self.Nsim+1))
        if multistage:
            Ypred = [0 for i in range(self.Nsim)]
            Wpred = [0 for i in range(self.Nsim)]
        else:
            Ypred = np.zeros((self.ny,self.prob_info['Np'],self.Nsim))

        if offset:
            Xss_sim = np.zeros_like(Xsim)   # steady state trajectory (controller)
            Yss_sim = np.zeros_like(Ysim)   # steady state output trajectory (controller)
            Uss_sim = np.zeros_like(Usim)   # steady state input trajectory (controller)

        ctime = np.zeros(self.Nsim)   # computation time
        Jsim = np.zeros(self.Nsim)    # cost/optimal objective value (controller)
        Yrefsim = np.zeros((self.nyc,self.Nsim+1))  # output reference/target (as sent to controller)
        Feas = np.zeros(self.Nsim)
        if CEM:
            CEMsim = np.zeros((1,self.Nsim+1))  # CEM trajectory
            CEM_stop_time = self.Nsim+1

        # set initial states and reset controller for consistency
        Xsim[:,0] = np.ravel(self.x0)
        Xhat[:,0] = Xsim[:,0]
        if observer is not None:
            observer.xhat = Xhat[:,0]
            observer.dhat = Dhat[:,0]
        Ysim[:,0] = np.ravel(self.hp(Xsim[:,0],Vsim[:,0]).full())

        if mpc_controller:
            c.reset_initial_guesses()

        # loop over simulation time
        CEM_stop_trigger = False
        for k in range(self.Nsim):
            startTime = time.time()

            Yrefsim[:,k] = self.myref(k*self.ts)
            if mpc_controller:
                if CEM:
                    if multistage:
                        c.set_parameters([Xhat[:,k], Yrefsim[:,k], CEMsim[:,k], Wset])
                    else:
                        c.set_parameters([Xhat[:,k], Yrefsim[:,k], CEMsim[:,k]])
                else:
                    if offset:
                        c.set_parameters([Xhat[:,k], Yrefsim[:,k], Dhat[:,k]])
                    else:
                        c.set_parameters([Xhat[:,k], Yrefsim[:,k]])
                res, feas = c.solve_mpc(warm_start=self.prob_info['warm_start'])
                Uopt = res['U']
                Jopt = res['J']
                if multistage:
                    Ypred[k] = res['Y'] # todo: add functionality to other controllers, then move this out of conditional statement
                    Wpred[k] = res['wPred']
            else:
                Jopt = np.nan

                if CEM:
                    in_vec = np.concatenate((Xhat[:,k], CEMsim[:,k]))
                else:
                    in_vec = np.concatenate((Xhat[:,k], Yrefsim[:,k]))
                Uopt = np.ravel((c.netca(in_vec)).full())
                Uopt = np.maximum(np.minimum(Uopt, self.prob_info['u_max']), self.prob_info['u_min'])

            ctime[k] = time.time() - startTime
            if mpc_controller:
                if not feas:
                    print('Was not feasible on iteration {} of simulation'.format(k))

            if offset:
                Xss_sim[:,k] = res['Xss']
                Uss_sim[:,k] = res['Uss']
                Yss_sim[:,k] = res['Yss']

            Usim[:,k] = np.ravel(Uopt)
            Jsim[k] = Jopt

            # send optimal input to plant/real system
            Xsim[:,k+1] = np.ravel(self.fp(Xsim[:,k],Usim[:,k],Wsim[:,k]).full())
            Ysim[:,k+1] = np.ravel(self.hp(Xsim[:,k+1],Vsim[:,k+1]).full())
            if CEM:
                CEMsim[:,k+1] = CEMsim[:,k] + np.ravel(self.prob_info['CEMadd'](Ysim[:,k+1]).full())
                if not CEM_stop_trigger and CEMsim[:,k+1] > Yrefsim[:,k]:
                    CEM_stop_time = k+1
                    CEM_stop_trigger = True
                if CEM_stop_trigger:
                    break

            # get estimates
            if observer is not None:
                xhat, dhat = observer.update_observer(Usim[:,k], Ysim[:,k+1])
                Xhat[:,k+1] = np.ravel(xhat)
                Dhat[:,k+1] = np.ravel(dhat)
            else:
                Xhat[:,k+1] = Xsim[:,k+1]

        Yrefsim[:,k+1] = self.myref((k+1)*self.ts)
        # create dictionary of simulation data
        sim_data = {}
        sim_data['Xsim'] = Xsim
        sim_data['Ysim'] = Ysim
        sim_data['Usim'] = Usim
        sim_data['Jsim'] = Jsim
        sim_data['Wsim'] = Wsim
        sim_data['Vsim'] = Vsim
        sim_data['Yrefsim'] = Yrefsim
        sim_data['ctime'] = ctime
        sim_data['Xhat'] = Xhat
        sim_data['Dhat'] = Dhat
        sim_data['Ypred'] = Ypred
        sim_data['feas'] = Feas
        if offset:
            sim_data['Xss_sim'] = Xss_sim
            sim_data['Uss_sim'] = Uss_sim
            sim_data['Yss_sim'] = Yss_sim
        if multistage:
            sim_data['wPred'] = Wpred
        if CEM:
            sim_data['CEMsim'] = CEMsim
            sim_data['CEM_stop_time'] = CEM_stop_time

        return sim_data
```

Due to the organization of the code into objects, any control policy object can be transferred to an experiment. In other words, an experiment involves the same three components described in Section B.1, where the control policy is the same as before; the system is the testbed (no code necessary); and the environment is a real-time experiment. The control policy can be moved over to the experiment, and all that is necessary to define is related to the environment, i.e., the experimental environment code and parameters, which can be modeled after the simulation code above. An example implementation is shown below and was taken from the code used for [76].

```
# experiment functions
#
# This file defines an Experiment class to be used for real time experiments on
# the atmospheric pressure plasma jet (APPJ) of model predictive
    controllers
# (MPCs) generated via the controller subclasses defined in controllers
    .py
#
# Requirements:
# * Python 3
#
# Copyright (c) 2022 Mesbah Lab. All Rights Reserved.
# Kimberly Chan
#
# This file is under the MIT License. A copy of this license is
    included in the
# download of the entire code package (within the root folder of the
    package).

## import 3rd party packages
import sys
sys.dont_write_bytecode = True
import numpy as np
import time
from datetime import datetime
import os

## import user functions
import KCutils.APPJPythonFunctions as appj

def ctok(T):
    """
    Function to convert from Celsius to Kelvin.
    """
    return T+273.15

class Experiment():
    """
    The Experiment class is used to create a wrapper for real-time
    experiments
    using the APPJ.
    """

    def __init__(self, Nsim, saveDir=os.getcwd(), name=None):
        """
        Instantiation of the Experiment class requires the input
    arguments
        Nsim, which denotes the length of the experimental run; saveDir
        (optional), which is a path to a particular save location; and
    name
        (optional) which is an additional identifier of the data from
    this
        class.
        """
        super(Experiment, self).__init__()
        self.Nsim = Nsim
        self.prob_info = None
        self.rand_seed = None

        self.saveDir = saveDir
        if not os.path.exists(saveDir+"Backup/"):
            self.backupSaveDir = saveDir+"Backup/"
            os.makedirs(saveDir+"Backup", exist_ok=True)
        print('\n\nBackup data will be saved in the following directory:')
        print(self.backupSaveDir)
        self.count = 0
        self.name = name
        if self.name is None:
            self.exp_name = 'Experiment_'+str(self.count)
        else:
            self.exp_name = self.name+'_Experiment_'+str(self.count)

        self.ol_count = 0

    def load_prob_info(self, prob_info):
        """
        This method loads the relevant problem information for experiment and
        assigns them as attributes of the class from the prob_info dict used by
        other classes included in this package.
        """
        # extract and save relevant problem information
        self.prob_info = prob_info

        # system sizes
        self.nu = prob_info['nu']
        self.nx = prob_info['nx']
        self.ny = prob_info['ny']
        self.nyc = prob_info['nyc']

        self.xss = prob_info['xss']
        self.uss = prob_info['uss']
        self.u_min = prob_info['u_min']
        self.u_max = prob_info['u_max']

        self.Np = prob_info['Np'] # prediction horizon
        self.x0 = prob_info['x0'] # initial state/point
        # self.y0 = prob_info['y0'] # initial outputs/measurements
        # self.u0 = prob_info['u0'] # startup inputs
        self.myref = prob_info['myref'] # reference function for the controlled output
        self.ts = prob_info['ts'] # simulation sampling time

    def run_closed_loop_mpc(self, c, ioloop, model=None, observer=None,
                            runOpts=appj.RunOpts(), devices=None, prevTime=0.0,
                            CEM=True, hw_flag=True, dnn_flag=False,
                            I_NORM=1e-4):
        """
        This method runs a closed-loop experiment of the APPJ using information
        derived from loading problem information and using an explicit MPC
        (defined by solving an optimal control problem (OCP) at each step). The
        problem information must be loaded before a closed-loop simulation can
        occur. The argument c is a MPC controller object created using one of
        the classes defined in the controller.py.
        """
        # check to ensure problem data has been loaded
        if self.prob_info is None:
            print('Problem data not loaded. Please load the appropriate problem data by running the load_prob_info method.')
            raise

        # get devices
        if devices is None:
            print('Device information not given! Please provide device info.')
            raise
        else:
            # serial device representation of Arduino
            key = 'arduinoPI'
            if key in devices:
                arduinoPI = devices[key]
            else:
                arduinoPI = None
                print(f'WARNING: {key} not in devices dict! Code will error...')
            # Arduino address
            key = 'arduinoAddress'
            if key in devices:
                arduinoAddress = devices[key]
            else:
                arduinoAddress = None
                print(f'WARNING: {key} not in devices dict! Code will error...')
            # Spectrometer
            key = 'spec'
            if key in devices:
                spec = devices[key]
            else:
                spec = None
                print(f'WARNING: {key} not in devices dict! Code will error...')
            # Oscilloscope
            key = 'instr'
            if key in devices:
                instr = devices[key]
            else:
                instr = None
                print(f'WARNING: {key} not in devices dict! Code will error...')

        ## get data sizes
        tasks, runTime = ioloop.run_until_complete(appj.async_measure(arduinoPI, prevTime, instr, spec, runOpts))
        if runOpts.collectData:
            thermalCamOut = tasks[0].result()
            Ts0 = thermalCamOut[0]
            specOut = tasks[1].result()
            I0 = specOut[0]*I_NORM
            oscOut = tasks[2].result()
            arduinoOut = tasks[3].result()
            outString = "Measured Outputs: Temperature: %.2f, Intensity: %.2f" % (Ts0, I0)
            print(outString)
        else:
            Ts0 = 37.0
            I0 = 100.0

        ## get controller type:
        mpc = False
        if hw_flag or dnn_flag:
            pass
        else:
            mpc = True
            Wset = self.prob_info['Wset']

        ## Instantiate container variables for storing experimental data
        if runOpts.collectData:
            if runOpts.saveData:
                Tsave = np.empty((self.Nsim,))
                Isave = np.empty((self.Nsim,))
                Psave = np.empty((self.Nsim,))
                qSave = np.empty((self.Nsim,))
                CEMsave = np.empty((self.Nsim))
                badTimes = []
            if runOpts.saveSpatialTemp:
                Ts2save = np.empty((self.Nsim,))
                Ts3save = np.empty((self.Nsim,))
            if runOpts.saveSpectra:
                if specOut is not None:
                    waveSave = np.empty((self.Nsim,len(specOut[2])))
                    specSave = np.empty_like(waveSave)
                    meanShiftSave = np.empty((self.Nsim,))
                else:
                    print('Intensity Data not collected! Entire spectrum will not be saved.')
                    runOpts.saveSpectra = False
            if runOpts.saveOscMeas:
                if oscOut is not None:
                    oscSave = np.empty((self.Nsim,len(oscOut)))
                else:
                    print('Oscilloscope data not collected! Nothing to save.')
                    runOpts.saveOscMeas = False
            if runOpts.saveEmbMeas:
                if arduinoOut is not None:
                    ArdSave = np.empty((self.Nsim,len(arduinoOut)))
                else:
                    print('Arduino Data not collected! Nothing to save.')
                    runOpts.saveEmbMeas = False
        # additional containers for system operation (controller, observer)
        Xhat = np.zeros((self.nx,self.Nsim+1))    # state estimation
        ctime = np.zeros(self.Nsim)   # computation time
        Yrefsim = np.zeros((self.nyc,self.Nsim))  # output reference/target (as sent to controller)
        CEMsim = np.zeros((self.nyc,self.Nsim+1)) # CEM accumulation
        if mpc:
            Jsim = np.zeros(self.Nsim)    # cost/optimal objective value (controller)
            Ypred = np.zeros((self.ny,self.Nsim,self.Np))
            Feasibility = np.zeros(self.Nsim) # feasibility of OCP

        # set initial states and reset controller for consistency
        Xhat[:,0] = np.ravel([Ts0,I0]-self.xss)
        CEMsim[:,0] = np.zeros((self.nyc,))
        count = 0

        # loop over simulation time
        if mpc:
            c.reset_initial_guesses()
        if CEM:
            CEM_stop_time = self.Nsim

        for k in range(self.Nsim):
            startTime = time.time()
            iterString = f'\nIteration {k} out of {self.Nsim}'
            print(iterString)

            ## Get measurements
            tasks, runTime = ioloop.run_until_complete(appj.
    async_measure(arduinoPI, prevTime, instr, spec, runOpts))
            # Temperature
            if runOpts.collectData:
                thermalCamMeasure = tasks[0].result()
                if thermalCamMeasure is not None:
                    Ts = thermalCamMeasure[0]
                    Ts2 = thermalCamMeasure[1]
                    Ts3 = thermalCamMeasure[2]
                else:
                    print('Temperature data not collected! Thermal
    Camera measurements will be set to -300.')
                    Ts = -300
                    Ts2 = -300
                    Ts3 = -300
                # Intensity
                specOut = tasks[1].result()
                if specOut is not None:
                    totalIntensity = specOut[0]*I_NORM
                    intensitySpectrum = specOut[1]
                    wavelengths = specOut[2]
                    meanShift = specOut[3]
                else:
                    print('Intensity data not collected! Spectrometer
    outputs will be set to -1.')
                    totalIntensity = -1
                    intensitySpectrum = -1
                    wavelengths = -1
                    meanShift = -1

                outString = "Measured Outputs: Temperature: %.2f,
    Intensity: %.2f" % (Ts, totalIntensity)
                print(outString)

                ## Save measurements to containers
                if runOpts.saveData:
                    Tsave[k] = Ts
                    Isave[k] = totalIntensity
                if runOpts.saveSpatialTemp:
                    Ts2save[k] = Ts2
                    Ts3save[k] = Ts3
                # Intensity spectra
                if runOpts.saveSpectra:
                    waveSave[k,:] = np.ravel(wavelengths)
                    specSave[k,:] = np.ravel(intensitySpectrum)
                    meanShiftSave[k] = meanShift
                # Oscilloscope
                if runOpts.saveOscMeas:
                    oscOut = tasks[2].result()
                    oscSave[k,:] = np.ravel(oscOut)
                # Embedded Measurements from the Arduino (note: several embedded measurements are not fully functional as of 2020/12)
                arduinoOut = tasks[3].result()
                prevTime = arduinoOut[0]
                if runOpts.saveEmbMeas:
                    ArdSave[k,:] = np.ravel(arduinoOut)
            else:
                Ts = 37.0
                totalIntensity = 100.0
            # measurements are collected after a control input has been applied
            Yrefsim[:,k] = self.myref(k)
            if k>0:
                Xhat[:,k] = np.ravel(np.asarray([[Ts,totalIntensity]])-self.xss)
                if CEM:
                    CEMsim[:,k] = CEMsim[:,k-1] + np.ravel(self.prob_info['CEMadd'](Ts).full())
                    if CEMsim[:,k]>Yrefsim[:,k]:
                        if CEMsim[:,k-1]<Yrefsim[:,k-1]:
                            CEM_stop_time = k
                        count+=1
                        if count > 3:
                            break

            ## Compute control input
            ctrl_stime = time.time()
            if mpc:
                print(Xhat[:,k])
                print(CEMsim[:,k])
                c.set_parameters([Xhat[:,k], Yrefsim[:,k], CEMsim[:,k], Wset])
                res, feas = c.solve_mpc(warm_start=self.prob_info['warm_start'])
                print(res['U'])
                print(feas)
                Uopt = np.asarray(res['U'])
                Jopt = res['J']
            else:
                if CEM:
                    x_in = np.concatenate((Xhat[:,k], CEMsim[:,k]))
                else:
                    x_in = np.concatenate((Xhat[:,k], Yrefsim[:,k]))
                Uopt = np.ravel((c.netca(x_in)).full())
                print(Uopt)

            Uopt[0] = np.clip(Uopt[0], self.u_min[0], self.u_max[0])
            Uopt[1] = np.clip(Uopt[1], self.u_min[1], self.u_max[1])
            Uopt = np.ravel(Uopt.T+self.uss)
            print(Uopt)
            ctrl_etime = time.time()
            ctime[k] = ctrl_etime - ctrl_stime

            powerIn = float(Uopt[0])
            flowIn = float(Uopt[1])
            ## Send optimal inputs to APPJ
            appj.sendControlledInputsArduino(arduinoPI, powerIn, flowIn
, arduinoAddress)

            # save inputs
            if runOpts.saveData:
                Psave[k] = np.ravel(Uopt[0])
                qSave[k] = np.ravel(Uopt[1])
            if mpc:
                Jsim[k] = Jopt
                # Ypred[:,:,k] = res['Y']
                # Wpred[k] = res['wPred']

            # Pause for the duration of the sampling time to allow the
system to evolve
            endTime = time.time()
            runTime = endTime-startTime
            print('Total Runtime was:', runTime)
            pauseTime = self.ts - runTime
            if pauseTime>0:
                print("Pausing for {} seconds...".format(pauseTime))
                time.sleep(pauseTime)
            else:
                print('WARNING: Measurement Time was greater than
Sampling Time! Data may be inaccurate.')
                if runOpts.saveData:
                    badTimes += [k]

        # shut off plasma
        appj.sendInputsArduino(arduinoPI, 0.0, 0.0, 100.0,
arduinoAddress)

        # create dictionary of experimental data
        exp_data = {}
        exp_data['Tsave'] = Tsave
        exp_data['Isave'] = Isave
        exp_data['Psave'] = Psave
        exp_data['qSave'] = qSave
        exp_data['Yrefsim'] = Yrefsim
```

```python
        exp_data['ctime'] = ctime
        exp_data['Xhat'] = Xhat
        exp_data['badTimes'] = badTimes
        if mpc:
            exp_data['Jsim'] = Jsim
            exp_data['Feasibility'] = Feasibility
            exp_data['Ypred'] = Ypred
        if CEM:
            exp_data['CEMsim'] = CEMsim
            exp_data['CEM_stop_time'] = CEM_stop_time
        if runOpts.collectSpatialTemp:
            exp_data['Ts2save'] = Ts2save
            exp_data['Ts3save'] = Ts3save
        if runOpts.collectEntireSpectra:
            exp_data['waveSave'] = waveSave
            exp_data['specSave'] = specSave
            exp_data['meanShiftSave'] = meanShiftSave
        if runOpts.collectOscMeas:
            exp_data['oscSave'] = oscSave
        if runOpts.collectEmbedded:
            exp_data['ArdSave'] = ArdSave

        # save experimental data to have a backup copy
        self.exp_data = exp_data
        np.save(self.backupSaveDir+self.exp_name+'.npy', exp_data)

        # save csv version of experimental data
        exp_saveDir = self.saveDir+self.exp_name+'/'
        if not os.path.exists(exp_saveDir):
            os.makedirs(exp_saveDir, exist_ok=True)
        exp_data_saver(exp_data, exp_saveDir, self.exp_name, runOpts)

        # increment and update class attributes to prepare for additional experiments
        self.count += 1
        if self.name is None:
            self.exp_name = 'Experiment_'+str(self.count)
        else:
            self.exp_name = self.name+'_Experiment_'+str(self.count)

        return exp_data

    def run_open_loop(self, ioloop, power_seq=None, flow_seq=None, runOpts=appj.RunOpts(), devices=None, prevTime=0.0):
        """
        This method runs a open-loop experiment of the APPJ using provided
        sequences of inputs.
        """
        # check for provided sequence of inputs
        if power_seq is None and flow_seq is None:
            print('Sequence of inputs not given! Please provide inputs.')
            quit()
        elif power_seq is None:
            P0 = float(input('Please enter a value for the power.\n'))
            flow_seq = np.asarray(flow_seq)
            power_seq = P0*np.ones_like(flow_seq)

        elif flow_seq is None:
            q0 = float(input('Please enter a value for the flow rate.\n'))
            power_seq = np.asarray(power_seq)
            flow_seq = q0*np.ones_like(power_seq)

        nP = len(power_seq)
        nq = len(flow_seq)

        if nP > nq:
            print('Sequence of POWER inputs longer than sequence of FLOW inputs. Using the shorter sequence...')
            Niter = nq
        elif nq > nP:
            print('Sequence of FLOW inputs longer than sequence of POWER inputs. Using the shorter sequence...')
            Niter = nP
        else:
            Niter = nP

        # unpack devices
        if devices is None:
            print('Device information not given! Please provide device info.')
            raise
        else:
            # serial device representation of Arduino
            key = 'arduinoPI'
            if key in devices:
                arduinoPI = devices[key]
            else:
                arduinoPI = None
                print(f'WARNING: {key} not in devices dict! Code will error...')
            # Arduino address
            key = 'arduinoAddress'
            if key in devices:
                arduinoAddress = devices[key]
            else:
                arduinoAddress = None
                print(f'WARNING: {key} not in devices dict! Code will
error...')
            # Spectrometer
            key = 'spec'
            if key in devices:
                spec = devices[key]
            else:
                spec = None
                print(f'WARNING: {key} not in devices dict! Code will
error...')
            # Oscilloscope
            key = 'instr'
            if key in devices:
                instr = devices[key]
            else:
                instr = None
                print(f'WARNING: {key} not in devices dict! Code will
error...')

        # initial measurement to get data sizes
        tasks, runTime = ioloop.run_until_complete(appj.async_measure(
arduinoPI, prevTime, instr, spec, runOpts))
        thermalCamOut = tasks[0].result()
        Ts0 = thermalCamOut[0]
        specOut = tasks[1].result()
        I0 = specOut[0]
        oscOut = tasks[2].result()
        arduinoOut = tasks[3].result()

        ## Instantiate container variables for storing experimental
data
        if runOpts.saveData:
            Tsave = np.empty((Niter,))
            Isave = np.empty((Niter,))
            badTimes = []
        if runOpts.saveSpatialTemp:
            Ts2save = np.empty((Niter,))
            Ts3save = np.empty((Niter,))
        if runOpts.saveSpectra:
            if specOut is not None:
                waveSave = np.empty((Niter,len(specOut[2])))
                specSave = np.empty_like(waveSave)
                meanShiftSave = np.empty((Niter,))
            else:
                print('Intensity Data not collected! Entire spectrum
will not be saved.')
                runOpts.saveSpectra = False
        if runOpts.saveOscMeas:
            if oscOut is not None:
                oscSave = np.empty((Niter,len(oscOut)))
            else:
```

```
            print('Oscilloscope data not collected! Nothing to save.')
            runOpts.saveOscMeas = False
    if runOpts.saveEmbMeas:
        if arduinoOut is not None:
            ArdSave = np.empty((Niter,len(arduinoOut)))
        else:
            print('Arduino Data not collected! Nothing to save.')
            runOpts.saveEmbMeas = False


    for i in range(Niter):
        startTime = time.time()
        print(f'\nIteration {i} out of {Niter}')

        # asynchronous measurement
        tasks, _ = ioloop.run_until_complete(appj.async_measure(arduinoPI, prevTime, instr, spec, runOpts))

        # Temperature
        thermalCamMeasure = tasks[0].result()
        if thermalCamMeasure is not None:
            Ts = thermalCamMeasure[0]
            Ts2 = thermalCamMeasure[1]
            Ts3 = thermalCamMeasure[2]
        else:
            print('Temperature data not collected! Thermal Camera measurements will be set to -300.')
            Ts = -300
            Ts2 = -300
            Ts3 = -300

        # Total intensity
        specOut = tasks[1].result()
        if specOut is not None:
            totalIntensity = specOut[0]
            intensitySpectrum = specOut[1]
            wavelengths = specOut[2]
            meanShift = specOut[3]
        else:
            print('Intensity data not collected! Spectrometer outputs will be set to -1.')
            totalIntensity = -1
            intensitySpectrum = -1
            wavelengths = -1
            meanShift = -1

        # Save measurements <--- takes on the order of 1-2e-5 seconds
        if runOpts.saveData:
```

```
                    Tsave[i] = Ts
                    Isave[i] = totalIntensity
                if runOpts.saveSpatialTemp:
                    Ts2save[i] = Ts2
                    Ts3save[i] = Ts3
                # Intensity spectra (row 1: wavelengths; row 2: intensities; row 3: mean value used to shift spectra)
                if runOpts.saveSpectra:
                    waveSave[i,:] = np.ravel(wavelengths)
                    specSave[i,:] = np.ravel(intensitySpectrum)
                    meanShiftSave[i] = meanShift
                # Oscilloscope
                if runOpts.saveOscMeas:
                    oscOut = tasks[2].result()
                    oscSave[i,:] = np.ravel(oscOut)
                # Embedded Measurements from the Arduino
                arduinoOut = tasks[3].result()
                prevTime = arduinoOut[0]
                if runOpts.saveEmbMeas:
                    ArdSave[i,:] = np.ravel(arduinoOut)

                print(f'Measured Outputs: Temperature: {Ts:.2f}, Intensity: {totalIntensity:.2f}\n')

                # Send inputs <--- takes at least 0.15 seconds (due to programmed pauses)
                # appj.sendInputsArduino(arduinoPI, power_seq[i], flow_seq[i], dutyCycle, arduinoAddress)
                appj.sendControlledInputsArduino(arduinoPI, float(power_seq[i]), float(flow_seq[i]), arduinoAddress)

                # Pause for the duration of the sampling time to allow the system to evolve
                endTime = time.time()
                runTime = endTime-startTime
                print('Total Runtime was:', runTime)
                pauseTime = runOpts.tSampling - runTime
                if pauseTime>0:
                    print(f'Pausing for {pauseTime} seconds...')
                    time.sleep(pauseTime)
                else:
                    print('WARNING: Measurement Time was greater than Sampling Time! Data may be inaccurate.')
                    if runOpts.saveData:
                        badTimes += [i]

            # shut off APPJ
            appj.sendInputsArduino(arduinoPI, 0.0, 0.0, 100.0, arduinoAddress)

```

```
        # create dictionary of experimental data
        exp_data = {}
        exp_data['Tsave'] = Tsave
        exp_data['Isave'] = Isave
        exp_data['Psave'] = power_seq
        exp_data['qSave'] = flow_seq
        exp_data['badTimes'] = badTimes
        if runOpts.collectSpatialTemp:
            exp_data['Ts2save'] = Ts2save
            exp_data['Ts3save'] = Ts3save
        if runOpts.collectEntireSpectra:
            exp_data['waveSave'] = waveSave
            exp_data['specSave'] = specSave
            exp_data['meanShiftSave'] = meanShiftSave
        if runOpts.collectOscMeas:
            exp_data['oscSave'] = oscSave
        if runOpts.collectEmbedded:
            exp_data['ArdSave'] = ArdSave

        # save experimental data to have a backup copy
        self.exp_data = exp_data
        np.save(self.backupSaveDir+'OL_data_'+str(self.ol_count)+'.npy'
    , exp_data)

        # save csv version of experimental data
        exp_saveDir = self.saveDir
        if not os.path.exists(exp_saveDir):
            os.makedirs(exp_saveDir, exist_ok=True)
        exp_data_saver(exp_data, exp_saveDir, 'OL_data_'+str(self.
    ol_count), runOpts)

        self.ol_count += 1
        return exp_data


def exp_data_saver(exp_data, saveDir, exp_name, runOpts):
    """
    This function saves experimental data generated using the
    Experiment class.
    This function is different from the automatic saving performed by
    the
    Experiment class when running an individual experiment. This
    function will
    save most data to csv files to make data easily interpretable
    without
    having to write a Python script to read the data.

    exp_data is the dictionary of experimental data obtained by running
     an
            experiment via the the Experiments class
    saveDir is the path to the save location
    timeStamp is the time stamp identifier of the series of experiments
    runOpts is a class that defines the run options used during the experiment
    """
    if runOpts.saveData:
        # extract data
        Tsave = exp_data['Tsave']
        Isave = exp_data['Isave']
        Psave = exp_data['Psave']
        qSave = exp_data['qSave']
        badTimes = exp_data['badTimes']

        dataHeader = "Ts (degC),I (a.u.),P (W),q (slm)"
        # Concetenate inputs and outputs into one numpy array to save it as a csv
        saveArray = np.hstack((Tsave.reshape(-1,1), Isave.reshape(-1,1), Psave.reshape(-1,1), qSave.reshape(-1,1)))
        np.savetxt( saveDir+exp_name+"_inputOutputData.csv", saveArray, delimiter=",", header=dataHeader, comments='')
        if badTimes:
            np.savetxt( saveDir+exp_name+"_badMeasurementTimes.csv", badTimes, delimiter=',')

    if runOpts.saveSpatialTemp:
        # extract data
        Tsave = exp_data['Tsave']
        Ts2save = exp_data['Ts2save']
        Ts3save = exp_data['Ts3save']

        dataHeader = "Ts (degC),Ts2 (degC),Ts3 (degC)"
        saveArray = np.hstack((Tsave.reshape(-1,1), Ts2save.reshape(-1,1), Ts3save.reshape(-1,1)))
        np.savetxt( saveDir+exp_name+"_dataCollectionSpatialTemps.csv", saveArray, delimiter=",", header=dataHeader, comments='')

    if runOpts.saveSpectra:
        # extract data
        waveSave = exp_data['waveSave']
        specSave = exp_data['specSave']
        meanShiftSave = exp_data['meanShiftSave']

        print("Entire spectra will be saved in a compressed .npz file with the following array variable names:\n"
              +"'wavelengths' for the range of wavelength values\n"
              +"'intensities' for the full intensity spectra corresponding to the wavelength range\n"
              +"'meanShifts' for the mean value used to shift the spectra.\n"
              +"Please use a Python script and numpy.load(file_name) to
    load this data.")
        np.savez_compressed( saveDir+exp_name+"_dataCollectionSpectra", wavelengths=waveSave, intensities=specSave, meanShifts=meanShiftSave)

    if runOpts.saveOscMeas:
        # extract data
        oscSave = exp_data['oscSave']

        dataHeader = "Vrms (V),Irms (A),Prms (W)"
        np.savetxt( saveDir+exp_name+"_dataCollectionOscilloscope.csv", oscSave, delimiter=",", header=dataHeader, comments='')

    if runOpts.saveEmbMeas:
        # extract data
        ArdSave = exp_data['ArdSave']

        dataHeader = "t_emb (ms),Isemb (a.u.),Vp2p (V),f (kHz),q (slm),x_pos (mm),y_pos (mm),dsep (mm),T_emb (K),P_emb (W),Pset (W),duty (%),V_emb (kV),I_emb (mA)"
        np.savetxt( saveDir+exp_name+"_dataCollectionEmbedded.csv", ArdSave, delimiter=",", header=dataHeader, comments='')

    print('\n\nData saved in the following directory:')
    print(saveDir)
    return

def process_experimental_data(exp_data, prob_info):
    pass
```

B.4 Run-to-Run Simulations and/or Experiments Facilitated by OOP

A crucial component of this dissertation involved run-to-run tuning/adaptation, i.e., repeated (but adapted) plasma treatments. To facilitate this, the code structure allows for the environment to be wrapped with a data-driven optimization, and simulations/experiments can proceed as defined in Algorithm 2.

The following code provides an example of code that can be used to perform Bayesian optimization real-time experiments. This code in particular is an example of how Ax [123] can be used to facilitate the additional layer of data-driven optimization. More detailed and complex implementations can be found in the code repositories listed in Appendix A.

```
'''
main script to run Bayesian optimization for plasma operating parameter tuning
on an atmospheric pressure plasma jet (APPJ) testbed

```

Algorithm 2 Basic algorithm to conduct iterative data-driven optimization.

```
1: Initialize GP(s); Define budget
2: for $n \leq$ budget do
3:     Update parameters of control policy
4:     Perform closed-loop simulation or real-time experiment
5:     Receive or calculate objective(s) $J_i(\theta_n)$ and constraint(s) $c_j(\theta_n)$
6:     Use $(\theta_n, \{J_i(\theta_n)\}_{i=1}^{m})$ and $(\theta_n, \{c_j(\theta_n)\}_{j=1}^{p})$ to update GPs
7:     Optimize acquisition function to get new parameters $\theta_{n+1}$
8:     $\theta_n \leftarrow \theta_{n+1}$
9: end for
```

```python
REAL-TIME EXPERIMENT

Requirements (main):
* Python 3
* PyTorch [https://pytorch.org]
* BoTorch [https://botorch.org] and Ax [https://ax.dev]
* Matplotlib [https://matplotlib.org] (for data visualization)
* a variety of other packages are necessary for communication with the testbed,
  please see the import list for these requirements

Copyright (c) 2024 Mesbah Lab. All Rights Reserved.

Author(s): Kimberly Chan

This file is under the MIT License. A copy of this license is included in the
download of the entire code package (within the root folder of the package).
'''
# import Python packages
import sys

sys.dont_write_bytecode = True
import os
from datetime import datetime

# import 3rd party packages
import numpy as np
import pandas as pd
import torch
from ax.service.ax_client import AxClient
from ax.core.arm import Arm
from ax.service.utils.instantiation import ObjectiveProperties
from ax.modelbridge.generation_strategy import GenerationStrategy, GenerationStep
from ax.modelbridge.registry import Models
from ax.modelbridge.factory import get_MOO_EHVI
from ax.modelbridge.modelbridge_utils import observed_hypervolume

SAMPLING_TIME = 1.0
NUM_OBJECTIVES = 3
OBJECTIVE_NAMES = ['alpha_H2O2', 'productivity', 'thermal dose']
MINIMIZE = False
OBJECTIVE_THRESHOLDS = [100.0, 100.0, 10.0]
n_steps = 2
PARAMETER_NAMES = [
    *[f"power{n}" for n in range(n_steps)],
    *[f"flow{n}" for n in range(n_steps)],
    *[f"distance{n}" for n in range(n_steps)],
]
PARAMETER_TYPES = ['range', 'range', 'range', 'range']
PARAMETER_BOUNDS = [[1.2, 1.8], [1.0, 1.5], [6.0, 8.0], [420, 600]]
PARAMETER_VALUE_TYPES = ['float', 'float', 'float', 'int']
NUM_BO_ITER = 20
grouped_data = True

init_data = pd.read_csv("./results/all_initial_data_r2_restricted.csv",
     index_col=0)
replicate_groups = [
    [f"sample{i}" for i in range(2, 4)],
    [f"sample{i}" for i in range(10, 13)],
    [f"sample{i}" for i in range(15, 18)],
    [f"paw_sample{i}" for i in range(1, 4)],
    [f"paw_sample{i}" for i in range(7, 10)],
]


if __name__ == "__main__":
    n = int(input("Enter BO replicate number:\n"))

    date = datetime.now().strftime("%Y_%m_%d_%H" + "h%M" + "m%S" + "s")
    os.makedirs(f"./results/{date}", exist_ok=True)
    mobo_save_filepath = f"./results/{date}/exp_ax_client_snapshot{n}.
    json"

    # define a function to evaluate the parameters for a given system;
    computes
    # the objectives and/or constraints to use as data for BO; often,
    this
    # encloses a single closed-loop simulation or real-time experiment
    def evaluate(parameters, no_noise=False):
        print("Parameters Suggested by BO:")
        print(parameters)

        r2_check = input("R2 >= 0.99? [Y/n]\n")
        if r2_check in ["y", "Y"]:
            pass
        else:
            repeat = input("repeated parameters? [Y/n]\n")
            if repeat:
                return "abandon"
            else:
                return "fail"

        objectives = {}
        while True:
            try:
                for i,obj_name in zip(range(NUM_OBJECTIVES),
    OBJECTIVE_NAMES):
                    obj_val = input(f"{obj_name} - Objective {i} value
    :\n")
                    if no_noise:
                        objectives[f"obj{i}"] = (float(obj_val), 0.0)
                    else:
                        objectives[f"obj{i}"] = (float(obj_val), None)
                break
            except Exception as e:
                print(e)
                pass
        return objectives

    ## Define the BO problem
    # define how to generate new candidate parameters (in this case all
    # candidate parameters are generated by BO)
    gs = GenerationStrategy(
        steps = [
            GenerationStep(
                model = Models.MOO,
                num_trials = -1,
                model_kwargs={
                    "fit_out_of_design": True,
                }
            )
        ]
    )

    # define the objective properties
    objectives = {
        f"obj{i}": ObjectiveProperties(
            minimize=MINIMIZE,
            threshold=OBJECTIVE_THRESHOLDS[i],
        ) for i in range(NUM_OBJECTIVES)
    }

    # define the parameter properties (also defines your search space)
    parameters = [
        {'name': p,
        'type': PARAMETER_TYPES[i],
        'bounds': PARAMETER_BOUNDS[i],
        'value_type': PARAMETER_VALUE_TYPES[i],
        } for i,p in enumerate(PARAMETER_NAMES)
    ]

    # create the AxClient that facilitates the BO process
    ax_client = AxClient(random_seed=42+n, generation_strategy=gs)
    ax_client.create_experiment(
        name='bo_plasma_chem',
        parameters=parameters,
        objectives=objectives,
        overwrite_existing_experiment=False,
        is_test=False,
    )

    # attach initial data
    # [...]

    # Perform BO iterations
    hv_list = []
    for i in range(NUM_BO_ITER):
        parameters, trial_index = ax_client.get_next_trial()
        raw_data = evaluate(
            parameters,
        )
        if type(raw_data) == str:
            if raw_data == "abandon":
                ax_client.abandon_trial(trial_index)
            elif raw_data == "fail":
                ax_client.log_trial_failure(trial_index)
            else:
                print("unknown data output")
                ax_client.log_trial_failure(trial_index)
        elif type(raw_data) == dict:
            ax_client.complete_trial(trial_index, raw_data=raw_data)
        else:
            print("unknown data output")
            ax_client.log_trial_failure(trial_index)

        # compute HV
        try:
            dummy_model = get_MOO_EHVI(
                experiment=ax_client.experiment,
                data=ax_client.experiment.fetch_data(),
            )
            hv_list.append(observed_hypervolume(modelbridge=dummy_model
))
```

```
            print(hv_list[-1])
        except Exception as e:
            print(e)
            print('Failed to compute hypervolume')
            hv_list.append(0.0)

        ax_client.save_to_json_file(mobo_save_filepath)
```