

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Automated Registration of Image Pairs with Dramatically Inconsistent Appearance

Permalink

<https://escholarship.org/uc/item/0nz6r315>

Author

Kwon, Youngwook Paul

Publication Date

2017

Peer reviewed|Thesis/dissertation

**Automated Registration of Image Pairs with Dramatically Inconsistent
Appearance**

by

Youngwook Paul Kwon

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sara McMains, Chair

Professor Alice M. Agogino

Professor Alexei A. Efros

Fall 2017

**Automated Registration of Image Pairs with Dramatically Inconsistent
Appearance**

Copyright 2017
by
Youngwook Paul Kwon

Abstract

Automated Registration of Image Pairs with Dramatically Inconsistent Appearance

by

Youngwook Paul Kwon

Doctor of Philosophy in Engineering - Mechanical Engineering

University of California, Berkeley

Professor Sara McMains, Chair

The objective of this research is to advance the state of the art in image matching algorithms, especially with regard to input image pairs that include dramatically inconsistent appearance (e.g., different sensor modalities, significant intensity/color changes, different times such as day/night and years apart, etc.). We denote this range of input as disparate input. To handle disparate input, one should be able to capture the underlying aspects not affected by superficial changes to appearance.

To this end, we present a novel image descriptor based on the distribution of line segments in an image; we call it DUDE (DUality Descriptor). By exploiting line-point duality, DUDE descriptors are computationally efficient and robust to unstable line segment detection. Our experiments show that DUDE can provide more true-positive correspondences for challenging disparate datasets.

Beyond traditional image matching, we have designed an effective autograding system for multiview engineering drawings that also uses DUDE to improve its performance. The autograding system needs to be able to compare drawings that may include appearance changes due to students' mistakes, but also needs to differentiate between allowable and erroneous translation and/or scale changes.

In the addition to hand-crafted descriptors, this research also investigates data-driven descriptors generated by new deep learning based approaches. Due to the lack of labeled disparate imagery datasets, it is still challenging to effectively target disparate input using deep learning approaches. Therefore we introduce an aggressive data augmentation strategy called Artificial Intensity Remapping (AIR). By applying AIR to standard datasets, one can obtain models that are more effective for registration of disparate data. Finally, we compare the DUDE descriptor to a deep learning based descriptor powered by AIR.

To my beloved parents

Contents

Contents	ii
List of Figures	v
List of Tables	ix
1 Introduction	1
2 Background and Previous Work	5
2.1 Feature detection and matching	5
2.2 Transformation model estimation	8
3 Linear Feature Matching Without Descriptors	11
3.1 Introduction	11
3.2 Algorithm	11
3.2.1 Line Segment Detection	11
3.2.2 Line Segment Merging	11
3.2.3 Transformation	16
3.2.4 Matching Score	17
3.2.5 Filtering	20
3.3 Results	20
3.4 Summary	32
4 Linear Feature Matching With DUDE Descriptor	33
4.1 Introduction	33
4.2 Proposed Method	33
4.2.1 Feature detection	34
4.2.2 Feature description	36
4.2.3 Feature Matching	40
4.3 Experimental Results	42
4.3.1 Evaluating detections	42
4.3.2 Evaluating descriptors	44
4.3.3 Shape matching using DUDE	46

4.3.4	Geometric characteristics of DUDE	48
4.4	Conclusion	50
5	Autograder for Multiview Drawing Using DUDE	52
5.1	Introduction	52
5.1.1	Missing and Incorrect Lines	54
5.1.2	Mismatched View Scales	54
5.1.3	Misaligned Views	54
5.1.4	Views in Incorrect Relative Locations	55
5.2	Related work	56
5.3	Algorithm	57
5.3.1	Single View Transformation Estimation	57
5.3.2	Application to Multiview Drawings	62
5.4	Grading Checks	62
5.4.1	Element Comparison	63
5.4.2	Front-Right View Alignment	63
5.4.3	Front-Top View Alignment	63
5.4.4	Uniform Scale	63
5.4.5	Mirroring	63
5.4.6	Rotation / Skew	65
5.5	Computational Improvement	65
5.5.1	Filtering	65
5.5.1.1	Attribute and Vertex Degree Filtering	65
5.5.1.2	Transformation Filtering	65
5.5.1.3	Connectivity Filtering	66
5.5.2	DUDE-based choice	66
5.6	Implementation Issues	68
5.6.1	Converting from DWG to DXF	69
5.6.2	Loading DXF in MATLAB	69
5.6.3	Merging Elements	69
5.6.4	Pre-defining Layer Names	69
5.7	Results	69
5.7.1	Grading result visualization	71
5.7.2	Comparison with human grading	71
5.8	Conclusion	73
6	Deep Learning Based Image Matching	77
6.1	Introduction	77
6.2	Datasets	78
6.3	Artificial Intensity Remapping	81
6.4	Experiments	83
6.4.1	Setup	83

6.4.2	Network training	83
6.4.3	Evaluation	85
6.5	Comparison with DUDE	87
6.6	Summary	88
7	Conclusion and Future Work	89
A	Histogram comparison	91
B	Objective loss functions	92
	Bibliography	93

List of Figures

1.1	Example image pairs with weakly consistent appearance: [a, left] different ranges; [a, right] different scales; [b, left] CT (Computed Tomography) and MR (Magnetic Resonance) [Kelman et al., 2007]; [b, right] Google Earth and SAR (Synthetic Aperture Radar) [Sandia National Laboratories, 2016]; [c] Somewhere in Livermore in 2004 and 2012; [d] Google Earth and Google Map.	2
1.2	Public datasets	3
2.1	Two types of Siamese network for image matching; (a) similarity: learning a vector embedding $D(x)$ for an input x , and (b) embedding: learning a real-valued similarity function $f(x_1, x_2)$ for an input pair x_1 and x_2	8
3.1	Line segment detector and Canny edge extractor (example 1).	12
3.2	Line segment detector and Canny edge extractor (example 2).	12
3.3	Line segment merging step.	13
3.4	The angular distance $\Delta\theta$ and the perpendicular distance Δd	13
3.5	Examples of the scores $Col(s, s')$ for six sample input pairs	14
3.6	Projection to 1D parametric space.	15
3.7	(a) and (b) show the detected lines using LSD. The coloring is just for visual help to distinguish individual line segments; (c) shows the 30 longest segments after the merging step.	15
3.8	The three line segments highlighted in red in the two images is the choice that yields the optimal transformation T^* . The three circles in each image indicate the intersection points of the three chosen segments. The circle color represents correspondence between the points.	18
3.9	(a) shows the registration result using the optimal transformation T^* . (b) visualizes how S and S' are finally matched.	19
3.10	Result comparison: (a) Our algorithm (b) Fedorov et al. [2003b]	19
3.11	Our input dataset.	24
3.12	Result: greentest	25
3.13	Result: circular_road	26
3.14	Result: mountain	27
3.15	Result: depth	28

3.16	Result: coast	29
3.17	Result: livermore1	30
3.18	Result: map	31
4.1	An example of two disparate images and their line segments: An EO (Electro-Optical) and a SAR (Synthetic Aperture Radar) image respectively from Sandia National Laboratories [2016] (top), initial line segments from LSD [Grompone von Gioi et al., 2012] (middle), and the proposed randomly merged line segments (bottom).	34
4.2	Merging criterion for the randomized merging process; [a] shortest distance (in pixel), [b] perpendicular distance (in pixel), and [c] angle (in degree) between line segment i and j .	35
4.3	Dual representation r, θ, f_1, f_2 of a line segment	37
4.4	An example of f -binning for a line segment with endpoints $f_1 = -0.4, f_2 = 0.1$ and six bins.	38
4.5	The summary of DUDE descriptor process.	38
4.6	2D visualization of DUDE	39
4.7	Three example DUDE descriptors	39
4.8	Line segment perturbation. In addition to the originally detected line segments, we intentionally duplicate each one d times while randomly perturbing its endpoints within ± 3 pixels. By doing so, we blur the DUDE descriptors and make them less sensitive to unstable endpoint detection.	40
4.9	Line segment coverage calculation. We sample points on s_1 and calculate the closest distance from each sample point to s_2 , and check if the distance is within a threshold. The coverage score between two segments, $c(s_1, s_2)$, is the fraction of the number of “covered” sample points (green) to the total number of sample points.	41
4.10	Threshold for our coverage metric $\mathbb{C}(S_i, S_j)$ (x -axis). Correct matches are shown in green, and incorrect matches are shown in red. Generally if $\mathbb{C}(S_i, S_j)$ is less than around 0.4, the match is highly likely incorrect.	41
4.11	Three examples of correctly matched features and relevant line segments. A red circle represents a feature, and green line segments are those within a feature.	42
4.12	Repeatability curves. Five detection systems are compared: SIFT(DoG), MSER, SYM-I, SYM-G, and MMID (ours). For a given image pair (left column), repeatability (y -axis) is computed when considering top- k (x -axis).	43
4.13	Precision(y)-Recall(x) curves comparing performance of descriptors: SIFT, SYM-D, JSPEC, DUDE, DUDE-F. The plots and example choices follow Bansal and Daniilidis [2013]. For each image pair (each column), and when using different feature detection types (each row), Precision-Recall curves are illustrated.	45

4.14	Matching of lines in the presence of unstable endpoint locations and noise: two sets of lines (a and b), and the matching result (c). Correspondences are shown both as the same number and color. Unmatched lines are shown in gray with no number.	48
4.15	Rotation (b) and scale (d) invariance: [b] Colors and numbers mark matching pairs; line segment 8 is the only one that is not matched correctly. [d] Both colors and thin gray lines mark matching pairs. Circles show histogram radius. Note that there are different numbers of line segments detected.	49
4.16	DUDE interpretations of the parallel and orthogonal line segments.	50
5.1	3D geometry represented in multiview drawings in Figure 5.2	52
5.2	An example of a formal multiview drawing. Note that in multiview engineering drawings the views are not labeled; the placement and alignment communicates the relative viewpoints.	53
5.3	Two classes of mistakes. Note that the labels on the views are not present in the actual multiview drawing.	54
5.4	Two more classes of mistakes. Note that the labels on the views are not present in the actual multiview drawing.	55
5.5	An example of (a) a solution drawing and (b) a student's drawings and (c) their naïve comparison. Because they have different scales, translations, and offsets, a naïve comparison does not work.	56
5.6	An example pair of views for transformation estimation.	58
5.7	(a) Even if we align the two views in terms of scale and translation, it is not easy to compare them at a glance; here half the elements still appear to be slightly off. (b) In fact, most elements match perfectly if the correct affine transformation is applied. The real problem is mirroring and two lines that only partially differ.	59
5.8	We estimate the transformation for each view individually using RANSAC. By applying the transformations to the views in D_s , we get the transformed version D'_s . Then the elements of D'_s and D_t can be compared one by one.	64
5.9	DUDE descriptor frame for a given point.	67
5.10	DUDE matching	67
5.11	Comparing the solution to a student's drawing. To be compared to D_t , all views in D_s are scaled 1.7 times larger. The top, front, and right views are translated $(-33.8, +186.2)$, $(-33.8, +39)$, and $(+187.7, +39)$, respectively. All views have zero rotation and skew. By aligning them, the algorithm finds incorrect and missing lines, which are represented in blue and dark red.	70
5.12	Color coded difficulty. The elements that are most frequently "missing" are shown in dark red, and those less frequently missing are shown in light red (a and c). The elements that are most frequently "incorrect" are shown in dark blue, and those less frequently incorrect shown in light blue (b). The numbers in the color bar indicate the fraction of student submissions that made the mistake for each element.	72

5.13	Comparison with human grading results.	73
5.14	Two examples of category B. Even though different rubrics are applied, errors are identified.	74
5.15	Two examples of category C. While a human grader failed to notice these mistakes, our autograding system found them.	75
5.16	An example of category D. Our algorithm failed to estimate an appropriate transformation for the front view of the solution drawing. Note that the student drawing (b) has multiple mistakes in the front view.	76
6.1	Examples of AIR: one original patch (leftmost) and its seven derived patches . . .	78
6.2	Examples of different datasets: Statue of Liberty (LY), Notre Dame (ND), Half Dome in Yosemite (YO) are from Brown et al. [2011]. Our disparate sets, MU1 and MU2, include images cropped from Hauagge and Snavely [2012] and others, and Razakarivony and Jurie [2015], respectively.	79
6.3	Joint distribution of corresponding pixel intensities. Hotter colors reflect more frequent occurrence. (a-e): Original patches (intensities $\in [0, 255]$). (f-j): Pre-processed patches (intensities $\in [-.3, .3]$).	80
6.4	Examples of AIR generation and application.	82
6.5	Training process of an example model training using LY and ND with a validation set from YO. The real curves are shown in light colors (light red for AUC and light blue for loss). To visualize their trends, we smooth them using exponential smoothing, and mark them in darker red and darker blue colors.	84
6.6	Precision-recall curves for each non-disparate test set.	86
6.7	Precision-recall curves for each disparate test set.	87

List of Tables

4.1	Repeatability	44
4.2	Descriptor mean average precision (mAP) evaluation and comparison	45
5.1	Computation time comparison.	68
6.1	Architectural comparison with the most closely related work: $C(w, s, n)$ denotes a convolution layer with n filters of size $w \times w$ and stride s ; $B(w, s, n)$ denotes $C(w, s, n)$ with batch normalization; $P(w)$ denotes a pooling layer of size $w \times w$ with stride w	82
6.2	AUC of precision-recall curves for cross-subset MVS test (bold denotes top performance).	86
6.3	AUC of precision-recall curves for disparate test (bold denotes top performance).	86
6.4	Mean average precision (mAP) evaluation and comparison between AIR-augmented deep-learning descriptor and DUDE. DUDE is comparable to AIR-augmented deep-learning descriptor when using SIFT as the detector, and a bit better in the case of a hypothetical perfect detector as represented by GRID.	88

Acknowledgments

I would like to express my deepest and sincere appreciation to my research adviser, Professor Sara McMains. I've always been fascinated with the intersection of mechanical engineering and computer science, and I feel grateful to have an adviser where our research interests are aligned. Professor McMains has been incredibly knowledgeable, supportive, and innovative in furthering my interest in both disciplines, and I've learned so much working with her on interesting and thought-provoking research projects. Her research acumen has been critical to my success, and throughout the process, she has been always one of the most kindest and patient advisers.

I would also like to thank Professor Alexei A. Efros and Professor Alice M. Agogino for being on my dissertation committee. Professor Efros and Professor Agogino have always been an inspiration to my work, and I've always admired their passion and meaningful contributions to society. I'm also grateful to the members of my qualifying exam committee including Professor David Dornfeld, Professor Dennis K. Lieu, and Professor Jonathan Shewchuk. They provided insightful suggestions to my research proposal.

In Summer 2012, I completed my first summer internship at Lawrence Livermore National Laboratory. My dissertation topic arose out of this experience, so it will always remain a pleasant memory. I want to thank Sheila Vaidya, a group director, for extending this research opportunity. I'm also grateful to Goran Konjevod, my mentor, was incredibly brilliant and supportive and Hyojin Kim, one of my co-workers for interesting and intellectual discussions.

In addition, I would like to express my appreciation towards my labmates, Xiang Li, Bodi Yuan, Zhongyin Hu, Sushrut Pavanaskar, Yusuke Yasui, Sushrut Pande, and Peter Cottle for enriching my life in Berkeley. I'll never forget the engaging conversations on research ideas, projects, papers, conferences, life at Berkeley, and beyond. The Korean Graduate Student Association also made my years at UC Berkeley the most memorable period in my life by fostering a sense of community and unforgettable friendships.

While I was finishing my dissertation, Phantom AI, an early stage autonomous driving startup, provided the unique opportunity to work on an exciting project marking my first contribution to the real world incorporating my research. I feel grateful to work with many talented and supportive team members, and it's been exciting to work with everyone towards the same goal.

My research was funded by National Science Foundation, Lawrence Livermore National Lab, and the Office of Teaching and Learning at UC Berkeley. Finally, I would like to thank the Mechanical Engineering and Computer Science departments for their supports including several scholarship opportunities.

Chapter 1

Introduction

Image matching is the task of finding a set of features in one image that can be identified as the same features in the counterpart image for a given input image pair. This task is one of the most important and fundamental tasks for diverse applications such as image stitching [Brown and Lowe, 2003], shape matching [Belongie et al., 2002], medical image registration [Stewart et al., 2003], and so on.

An extremely useful paradigm is to use local features. Specifically, feature-based methods typically require *feature detection* and *feature description* steps. A feature detector detects salient features (e.g., corner points, keypoints, or blob-like regions) in an image that are likely to be detected in the counterpart image. A feature descriptor encodes the local appearance of the detected feature (e.g., a sub-patch around a feature) with a vector of a finite size. Generally, detection and description are done individually for each image. Then, one can find matches based on descriptor similarity, which is defined by a certain similarity metric (e.g., Euclidean distance).

We can divide the difficulties encountered during image registration into two categories.

First, even though an image pair contains the same scene or object, the images may include range and/or scale changes as shown in Figure 1.1a. In other words, viewpoint changes may cause geometric transformations (e.g., affine transformations, or in a more general case, homographies). Due to the viewpoint changes, a naïve approach such as excerpting a sub-patch from image 1 with a fixed window size and comparing it within image 2 in a sliding window manner would not work. Much research has addressed the difficulty due to viewpoint changes, and therefore image feature matching is relatively robust for images taken with the same sensors under the same conditions but with only view point changes (e.g., image panorama stitching).

However, there are still many challenging images where existing methods fail. Difficulties often arise from weakly consistent appearance such as: images from different modalities, where the different modalities of sensors may even lead to some features being completely absent in one modality, or gradient reversals or other extreme changes in appearance (Figure 1.1b); images taken years apart (Figure 1.1c); drawings vs. photos (Figure 1.1d); etc. Automatically finding and matching correspondences between these images is still quite dif-

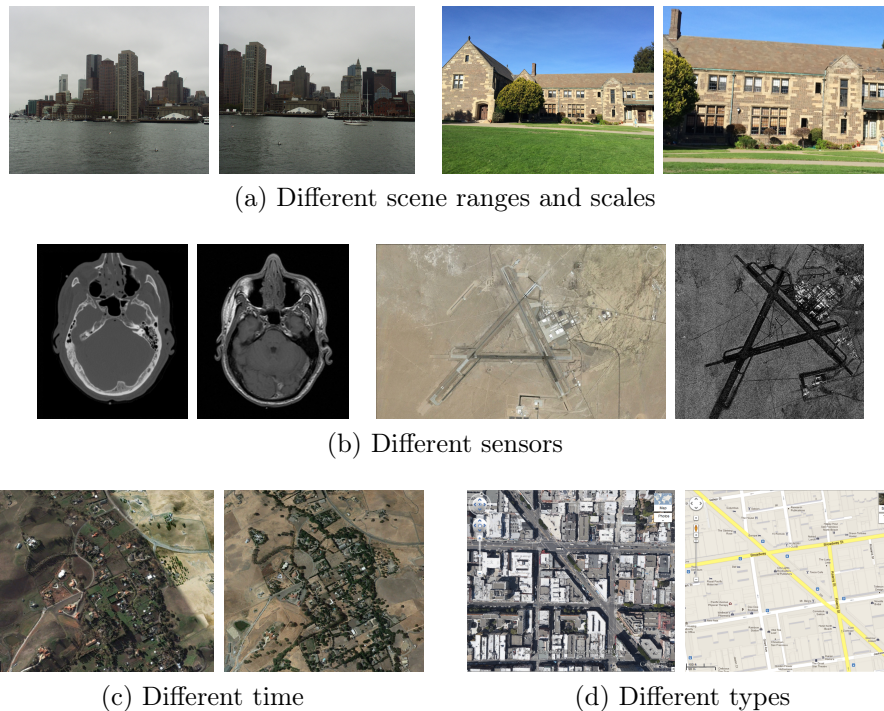


Figure 1.1. Example image pairs with weakly consistent appearance: [a, left] different ranges; [a, right] different scales; [b, left] CT (Computed Tomography) and MR (Magnetic Resonance) [Kelman et al., 2007]; [b, right] Google Earth and SAR (Synthetic Aperture Radar) [Sandia National Laboratories, 2016]; [c] Somewhere in Livermore in 2004 and 2012; [d] Google Earth and Google Map.

difficult. We denote the cause of these difficulties as *disparate* appearance.

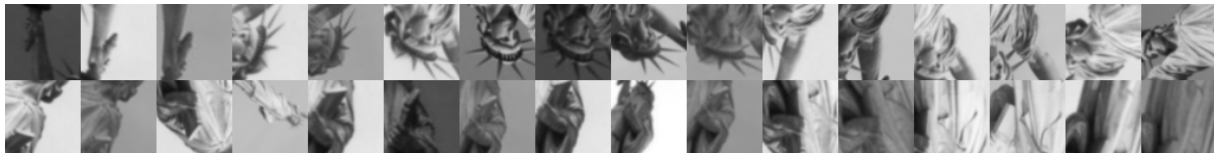
Let’s consider how the feature detection and feature description steps relate the two types of difficulties. To overcome view change difficulties, feature detectors are more critical to successful performance than feature descriptors, since it is important to *recognize* candidate regions of interest (ROIs) invariant to transformations that input images may have. If one can successfully find and normalize a good set of candidate ROIs, the corresponding ROIs would have similar appearance. So, feature descriptors may not play as important a role as feature detections.

In the case of disparate appearance, not only is detection challenging, but also descriptors need to capture the underlying aspects not affected by superficial changes to appearance. In other words, even with an almost perfect level of feature detection given, performance may still be poor. This dissertation focuses more on the latter challenge, and therefore more on descriptors than detectors.

To visually show the two types of difficulty, we give examples from some well-known and publicly accessible datasets in Figure 1.2. The Oxford dataset in Figure 1.2a and the Brown dataset in Figure 1.2b mainly cover the first category of view point changes, whereas the



(a) View point (including zoom and rotation) [Visual Geometry Group, University of Oxford, 2016]



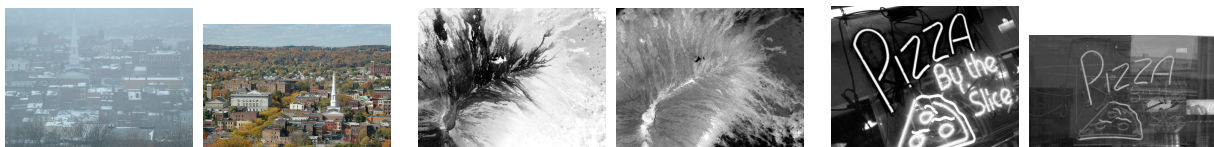
(b) View point [Brown et al., 2011]



(c) Illumination [Wang et al., 2011]



(d) Architecture (time, photo vs. drawing, and so on) [Hauagge and Snavely, 2012]



(e) Kelman (sensor, time, and so on) [Kelman et al., 2007]

Figure 1.2. Public datasets

illumination dataset in Figure 1.2c, the architecture dataset in Figure 1.2d, and the Kelman dataset in Figure 1.2e cover the second category of *disparate* appearance.

In considering the disparate appearance challenge, we consider two questions: (1) what information tends to be consistently preserved across different modalities despite significant appearance changes (e.g., reversal of brightness and darkness, partial absence of edges, etc.), and (2) how can we capture and describe the information effectively and efficiently without being affected by local appearance?

Inspired by the first question, we propose detecting (hundreds or thousands of) line segments from a given image and exploiting them as input to our descriptor, because lines will be preserved despite reversals of intensities. Since those detected line segments capture the shape structures in images — even when images do not include well-defined shapes with closed boundaries such as in aerial images — if one can describe a point in one image by

relying on its geometric position relative to (hundreds or thousands of) line segments, then the description can be consistent in the counterpart image. In Chapter 3, we investigate if line segments can be appropriate cues for image matching. We extract line segments from an image and use them for registration. We search for corresponding line segments and estimate the transformation using RANSAC [Fischler and Bolles, 1981]. Even though we show meaningful results using line segments, this method has a limitation in that the search space is huge. This limitation motivates the second question mentioned above.

To answer the second question, we propose a novel descriptor system, **DUDE** (DUality DEscription), that uses a histogram based on a weighted transform of lines to an r - θ dual space (or parameter space) of lines. By exploiting line-point duality, DUDE captures geometric relationships very efficiently. Our experiments show that the DUDE descriptor can be distinctive within an image and consistent across different modalities, which are the key characteristics of good descriptors. We introduce DUDE in Chapter 4.

DUDE is not limited to image matching. In Chapter 5, we introduce an application of DUDE to compare multi-view engineering drawings for grading, where solution and student drawings may differ in scale and translation. Using DUDE, we develop an educational autograding system that can differentiate between allowable transformations (i.e., distance between adjacent views) and student errors (i.e., distances between features in a single view).

So far, we have discussed mainly hand-crafted descriptors. More recently, approaches exploiting deep learning to generate descriptors show impressive performance [Simo-Serra et al., 2015; Zagoruyko and Komodakis, 2015; Kumar et al., 2016; Lin et al., 2016; Balntas et al., 2016; Yi et al., 2016]. Motivated by such trends, we propose deep-learning-based descriptors that are effective for disparate input in Chapter 6. This research is novel in that we achieve this not by learning from a target disparate-appearance dataset but from a non-disparate-appearance dataset.

Under this strategy, although the explicit goal is to propose a better descriptor embedding for disparate input, the key is closely related to better generalization. To learn effectively from a non-disparate-appearance dataset, we need to ensure that the learning process does not overfit to the training data, so that it will be robust to extremely diverse input. In many cases in machine learning research, evaluation of an algorithm is conducted by separating a single dataset into training and test sets. Despite this separation, the sets may reflect undesired and unintended correlation (e.g., a certain type of sensor), which will lead to the trained models being less robust for images in the wild.

For better generalization, we introduce a new data augmentation strategy, *Artificial Intensity Remapping* (AIR) in Chapter 6. Our experiments show that a model powered by AIR outperforms not only for standard (unimodal) datasets but also for disparate-appearance datasets. We compare our hand-crafted descriptor DUDE to multiple deep-learning based descriptors powered by AIR.

Prior to starting our journey, we provide an overview of the literature in the next chapter.

Chapter 2

Background and Previous Work

2.1 Feature detection and matching

Zitová and Flusser [2003] broadly divide feature matching techniques into area-based and feature-based approaches. Area-based methods focus more on feature matching than feature detection. Area-based methods usually do not attempt to detect salient objects in images, but try to find correspondences by comparing fixed certain windows (windows of a predefined size or even entire images) in the reference image with sliding windows in the other image. We can compare the two windows using correlation-like methods, Fourier methods, etc. Area-based methods are especially vulnerable to changes in images. For example, since even the shape of a window (usually a rectangle) in the reference image can be distorted by a transformation, then it may be naive to expect to find a good correspondence with the window of the original shape in the sensed image.

Due to the limitation of area-based approaches, much research has been focused on feature-based approaches, putting more emphasis on the feature detection and description stages, and therefore, making an effort to detect distinctive features. Feature descriptors and similarity measures for them are used for the matching stage.

Many image feature detectors and descriptors have been proposed. Examples of feature detectors include Harris corner detector [Harris and Stephens, 1988], LoG (Laplacian of Gaussian) [Lindeberg, 1998], DoG (Difference of Gaussian) [Lowe, 2004], Harris-affine [Mikolajczyk and Schmid, 2004], Hessian-affine [Mikolajczyk et al., 2005], and MSER (Maximally Stable Extremal Region) [Matas et al., 2002].

One of the most popular feature descriptors is SIFT [Lowe, 2004]. Mikolajczyk and Schmid [2005] conducted a performance evaluation of local descriptors, and SIFT showed the overall best results. Many other descriptors can be found in Wang et al. [2011]; Mikolajczyk and Schmid [2004]; Van De Sande et al. [2010].

These local feature detectors and descriptors have sophisticated designs to make them robust to scale, illumination, and large view angle changes. Nevertheless, they do not exhibit a similar level of repeatability and consistency across disparate images such as the ones we

are targeting.

Relatively less research has been devoted to matching and aligning these kinds of image pairs. [Stewart et al. \[2003\]](#) present an algorithm to find an overall transformation from a single correspondence, based on local descriptor distance and region growing. [Kelman et al. \[2007\]](#) build an experimental dataset of images including different modalities and extreme illumination changes, and present the SIFT-GM (Gradient Mirroring) and SIFT-GMEP (Gradient Mirroring and Edge Precursors) descriptors, which are variations of SIFT to adapt to different modalities. [Irani and Anandan \[1998\]](#) estimate the transformation (e.g., affine) parameters to register images with locally determined information. Their assumption is that when a pixel of one image approaches the correct location in the counterpart image, their local similarity (e.g., normalized-correlation) will be maximized and the similarity surface around the correct location will have a concave shape.

[Shechtman and Irani \[2007\]](#) propose Local Self-Similarities (LSS) that highlight intra-image changes. The strategy is successful in detecting patterns across different image properties (colors, textures, etc.), but it does not provide rotational invariance. Another approach is using dense descriptors. Typically these incur high costs in time and memory usage even just for encoding the descriptors. Matching two sets of dense descriptors also requires an optimization process. The dense descriptor DAISY [[Tola et al., 2010](#)] emphasizes its descriptor calculation efficiency, but it does not provide rotation invariance. DSIFT [[Vedaldi and Fulkerson, 2010](#)] does not provide scale invariance. Above all, using dense versions of a descriptor (e.g., DSIFT) with optimization techniques would be more useful when its sparse version (e.g., SIFT) guarantees a certain level of performance — for the purpose of acquiring higher numbers of dense feature matches (as in optical flow). Unfortunately, such performance has not been demonstrated on the challenging image pairs we target.

Compared to point and region features, line features have received less attention. [Schmid and Zisserman \[1997a\]](#) propose a method to match line segments across views (even with a wide baseline). However, the algorithm requires known epipolar geometry between the views. A line descriptor named MSLD (Mean-Standard deviation Line Descriptor) was proposed by [Wang et al. \[2009b\]](#). They define Pixel Support Regions (PSR) of line segments, subdivide the PSRs into sub-regions, and characterize the sub-regions using a SIFT-like strategy. Because the resulting descriptor sizes vary with line lengths, they normalize the descriptors using mean and standard deviation.

Line Signature [[Wang et al., 2009a](#)] by Wang et al. and Bunch Of Lines Descriptor (BOLD) [[Tombari et al., 2013](#)] by Tombari et al. create a descriptor for a line segment by clustering a small number (e.g., 10) of the nearest line segments into local groups, and then encoding pairwise geometric relationships between the line segment and each segment in the cluster. However, the geometric primitives determined by small numbers of line segment endpoints typically vary across image modalities (e.g., disconnected short segments). In addition, because the segment-pair-based geometric relationships can be computationally expensive, a line segment cluster typically consists of only a small number of line segments.

More successful disparate image matching is reported by [Hauagge and Snavely \[2012\]](#) and [Bansal and Daniilidis \[2013\]](#). [Hauagge and Snavely \[2012\]](#) present fea-

ture detection and descriptors based on local symmetries. They also present a challenging dataset, mostly architectural scenes that include symmetric shapes, exhibiting dramatic variations in lighting, time, modality, etc. Using the same dataset, [Bansal and Daniilidis \[2013\]](#) present a method analyzing the eigen-spectrum of the joint image graph constructed from all pixels in images, and achieve impressive experimental results. However, [Hauagge and Snavely \[2012\]](#) may not be suitable for images that do not include symmetric objects, and [Bansal and Daniilidis \[2013\]](#) is highly expensive both in memory and in time for eigen decomposition of a huge matrix, which is impractical for most images. We evaluate our method following their evaluation standards and show DUDE can achieve similar performance with more efficient computation.

A Siamese network [\[Bromley et al., 1993\]](#) is a general deep-learning (or neural) network for learning pairwise similarity, used in various applications including evaluating image patch similarity [\[Simo-Serra et al., 2015; Zagoruyko and Komodakis, 2015; Kumar et al., 2016; Lin et al., 2016; Yi et al., 2016\]](#), image retrieval [\[Qi et al., 2016\]](#), signature verification [\[Bromley et al., 1993\]](#), and face verification [\[Taigman et al., 2014\]](#). As a variant of Siamese networks, Triplet networks are used by [Balntas et al. \[2016\]](#).

Broadly there are two approaches in using Siamese networks: learning a descriptor embedding (Figure 2.1a) or learning pairwise similarity itself (Figure 2.1b). In both cases, there are two identical branches (“towers”) in the network that share exactly the same set of weights W . Training data consists of a set of patch pairs (x_1, x_2) with their binary labels $l \in \{0, 1\}$ reflecting whether a pair corresponds (true sample) or not (false sample).

In the embedding case, as in traditional descriptors, the resulting descriptors should work for L_2 (Euclidean) distance. In the similarity case, additional layer(s) (“metric network(s)”) that infer a real-value output from the output of the two towers can be seen as learning a metric as well. Although the similarity approach may show slightly better performance with its greater flexibility because it also learns its own descriptor distance metric, note that one needs to go through the full architecture to compare two patches. In contrast, in the embedding approach, one can calculate descriptors a priori, so that for any given pair, only an L_2 calculation is required.

Two state-of-the-art approaches are described in [Simo-Serra et al. \[2015\]](#) and [Kumar et al. \[2016\]](#). [Simo-Serra et al. \[2015\]](#) proposes an embedding architecture and showed promising results. [Kumar et al. \[2016\]](#) proposes an additional term called global loss in loss functions, and tested its efficacy using several combinations of both cases (embedding and similarity) and different architectures (Siamese and Triplet networks), and reported overall better performance with Triplet networks than Siamese networks. To compare [Simo-Serra et al. \[2015\]](#); [Kumar et al. \[2016\]](#) and our work, we limit the scope of this paper to Siamese networks for embedding. However, AIR is not limited to Siamese networks; AIR is compatible with any types of network or objective function as a data augmentation strategy.

Since deep neural networks need to be trained on a huge number of training images to achieve satisfactory performance, data augmentation can boost performance. Multiple combinations of horizontal/vertical flipping, cropping, color jittering, etc. are popularly used. In the object recognition research literature [\[Graham, 2014; Goodfellow et al., 2013; Lin](#)

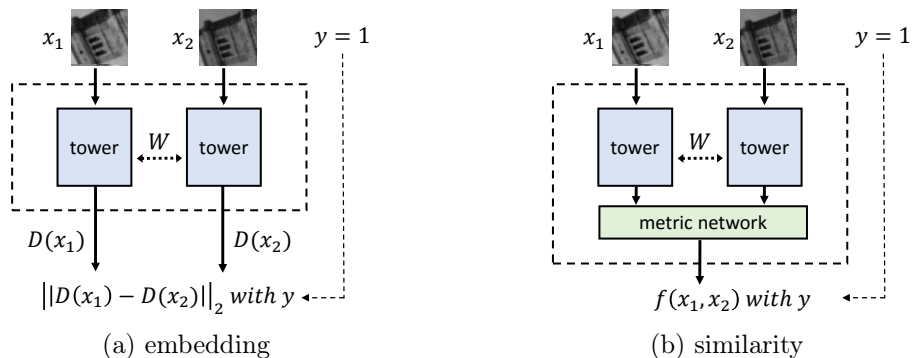


Figure 2.1. Two types of Siamese network for image matching; (a) similarity: learning a vector embedding $D(x)$ for an input x , and (b) embedding: learning a real-valued similarity function $f(x_1, x_2)$ for an input pair x_1 and x_2

et al., 2013; Springenberg et al., 2015], data augmentation strategies are typically limited to translation and horizontal flipping (vertical flipping is seldom used in object recognition, since vertically flipped objects such as cats and pedestrians rarely occur).

A more complex strategy can be found in Krizhevsky et al. [2012], which uses principal component analysis (PCA) to guide altering RGB channel values in training images. This strategy reduced the top-1 error rate by over 1% in the ImageNet 2012 competition. An open source library for deep learning computation, Tensorflow [Tensorflow TM, 2016], provides built-in functions to adjust the contrast, brightness, hue, and saturation of images for the purpose of data augmentation. However, these methods are not appropriate to augment data in descriptor embedding applications; this is because the PCA method [Krizhevsky et al., 2012] and the built-in Tensorflow functions add or multiply a (random) constant value to all pixels in a training image. In descriptor embedding, since mean and standard deviation normalization (to zero mean and unit variance) is a standard preprocessing step for the purpose of normalizing possible illumination or contrast changes, simply adjusting all pixels by adding or multiplying by a constant value does not have any effect after the normalization preprocessing.

2.2 Transformation model estimation

Our main purpose in finding correspondences is to *register* a given pair of images, which requires estimating a transformation between the images. In this section, we provide an overview of possible transformations.

There are various types of transformations such as similarity transforms, affine transforms, projective transforms, and transforms with higher degrees of freedom such as polynomial models.

Similarity transforms include translation, rotation, uniform scaling, and any compositions thereof. Using homogeneous coordinates, a similarity transform can be written as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s \cos \theta & -s \sin \theta & t_x \\ s \sin \theta & s \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.1)$$

where (x', y') is a corresponding point in the reference image, (x, y) is its corresponding point in the sensed image, θ is the rotation angle, (t_x, t_y) is the translation vector, and s is the scale factor. A non-rigid transformation that preserves distances after mapping can be regarded as a special case of the similarity transform with scale of 1. Under similarity transforms, shapes of objects are preserved.

Affine transformations are a superset of similarity transforms that also include scaling with different aspect ratios, reflection, skew, and compositions thereof. Sets of parallel lines remain parallel after an affine transformation. Using homogeneous coordinates, an affine transform can be written as

$$\begin{aligned} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} & (2.2) \\ &= \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & k & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, & (2.3) \end{aligned}$$

where s_x and s_y are the scale factors in the x and y axes respectively, and k is the skew factor. Because this model has six degrees of freedom, at least three non-collinear corresponding pairs are required to solve the system.

Projective transformations (called homography) can represent any mapping in projective space. Using homogeneous coordinates, a projective transformation can be written as

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.4)$$

Note that, because the projective transformation matrix is a homogeneous matrix, it has eight degrees of freedom. At least four linearly independent corresponding pairs are required to solve the system.

We restrict our research to these *global mappings* such as similarity, affine, and projective transformation, which can cover a large range of geometric transformations with a few parameters in a simple and global manner. For some applications, *local mappings* [Zitová and Flusser, 2003] are required, for example, if objects in the scene are flexible and have distortion, as in medical imaging applications. These types of transformation are beyond the scope of our current work.

In this section, we discussed various transformation assuming we have true corresponding points. Note that the feature detection and feature description steps provide only a set of (usually noisy) candidate corresponding points. There are several ways to filter false candidates such as RANSAC [[Fischler and Bolles, 1981](#)], which we will explain and apply in [Chapter 3](#) and [Chapter 5](#). Even though there exist such post-processes, it is still important to have good feature detectors and descriptors, since bad input to those post-processes can render the post-process results meaningless, and better input can yield better performance (speed and accuracy) of post-processes.

Chapter 3

Linear Feature Matching Without Descriptors

3.1 Introduction

In this chapter, we investigate if line segments can be appropriate cues for image matching. We extract line segments from an image and use them for registration. We search for corresponding line segments and estimate the image transformation using RANSAC [Fischler and Bolles, 1981].

3.2 Algorithm

3.2.1 Line Segment Detection

To detect line segments in images, we have chosen to use the Line Segment Detector (LSD) algorithm [Grompone von Gioi et al., 2012], while others use a Hough transform [Dubrofsky and Woodham, 2008; Habib and Alruzouq, 2004] or a Canny edge extractor [Coiras et al., 2000; Schmid and Zisserman, 1997b]. The Hough transform extracts not line segments but full lines in images, from which the line segments would need to be extracted. The Canny edge extractor [Canny, 1986] requires a threshold parameter δ , and the detected edges are not guaranteed to be linear. The LSD algorithm detects line segments and runs in linear time without parameter tuning. Figure 3.1 and 3.2 show the results of the LSD compared to the Canny edge extractor.

3.2.2 Line Segment Merging

The line segments detected using LSD are incomplete input for us in the sense that a single salient and useful line segment may be detected as a few broken and/or overlapped segments. To deal with this issue, we add a merging step. Figure 3.3 illustrates the results of our

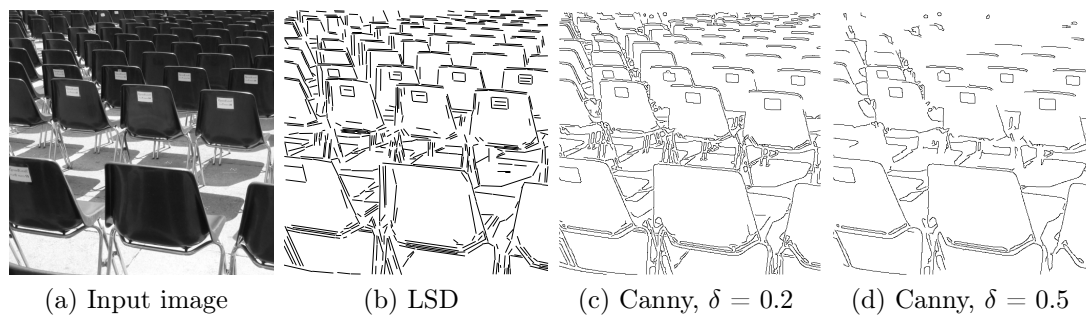


Figure 3.1. Line segment detector and Canny edge extractor (example 1).

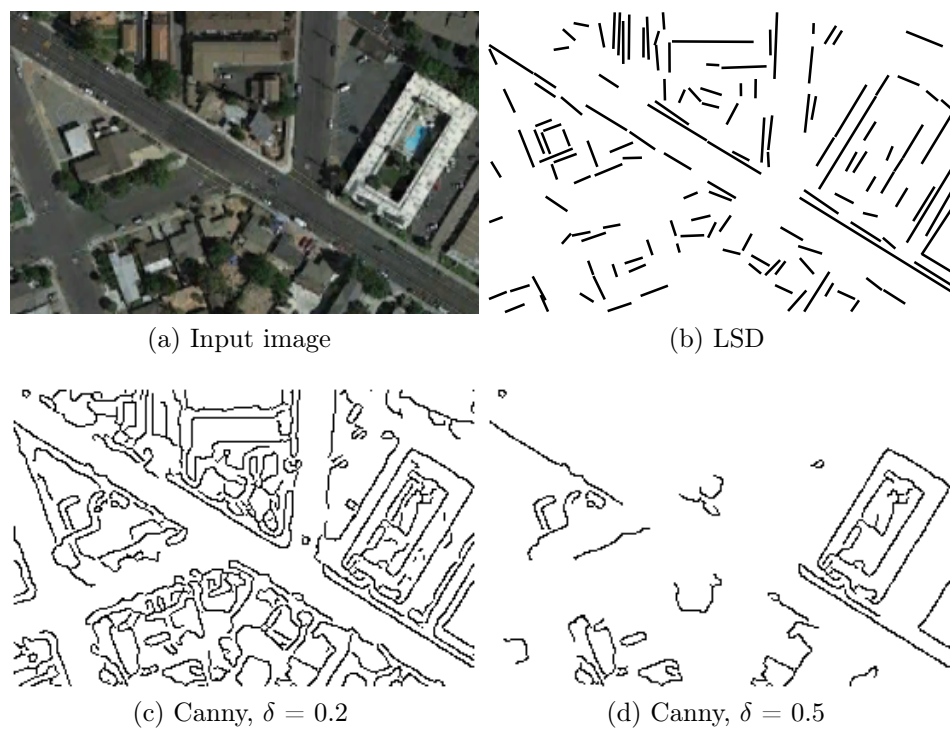


Figure 3.2. Line segment detector and Canny edge extractor (example 2).

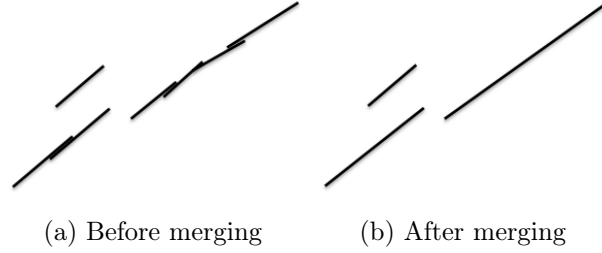
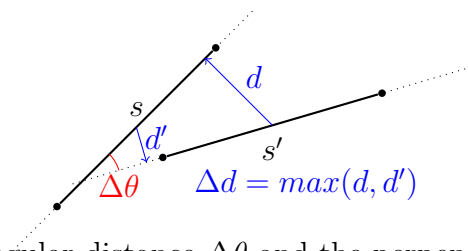


Figure 3.3. Line segment merging step.

Figure 3.4. The angular distance $\Delta\theta$ and the perpendicular distance Δd

procedure. We merge two line segments if they are (i) collinear (within a threshold) and (ii) overlapped. We describe as follows how to check the collinearity and overlap for two given line segments s and s' .

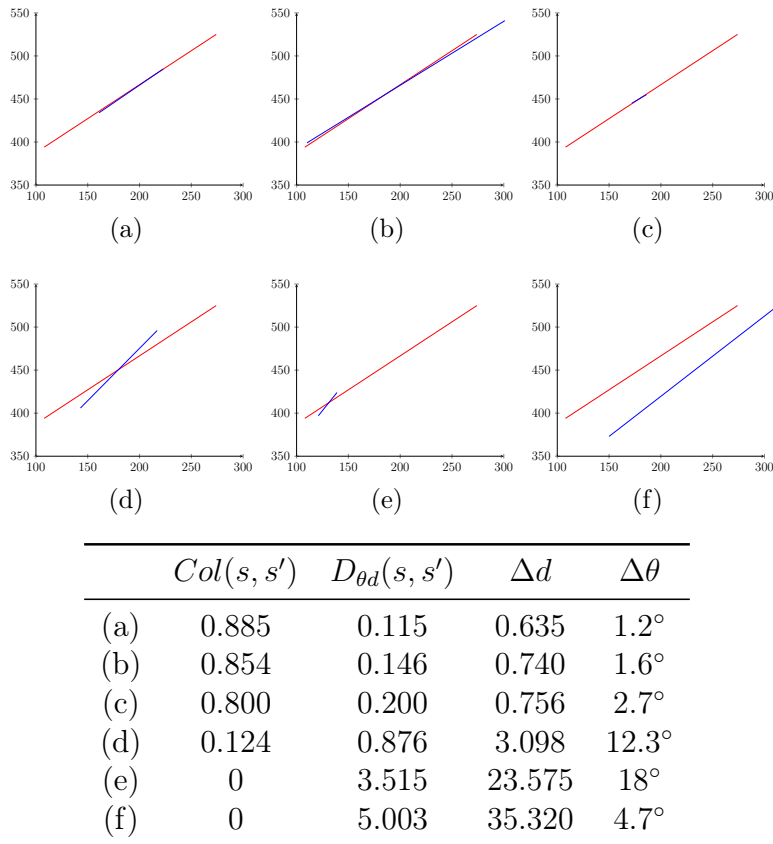
First, we define a distance function $D_{\theta d}(s, s')$ over two line segments s and s' . This definition is based on that of Coiras et al. [2000], but modified from a non-symmetric distance to a symmetric distance. As illustrated in Figure 3.4, for two given line segments s and s' , we can calculate the angular difference $\Delta\theta$, and the perpendicular distance Δd , where $\Delta\theta$ is the acute angle that s and s' (or their extended lines) make, and Δd is the maximum of two perpendicular distances: the distance from the midpoint of s to the line containing s' (d' in Figure 3.4), and the distance from the midpoint of s' to the line containing s (d in Figure 3.4). Then we define the distance function $D_{\theta d}(s, s')$ between the two line segments s and s' as:

$$D_{\theta d}(s, s') = \frac{1}{\sqrt{2}} \sqrt{\left(\frac{\Delta\theta}{\theta_\delta}\right)^2 + \left(\frac{\Delta d}{d_\delta}\right)^2}, \quad (3.1)$$

where θ_δ and d_δ are thresholds for angular difference and perpendicular distance, respectively. We set $\theta_\delta = 5^\circ$, $d_\delta = 5$. For example, if s and s' are collinear, then $\Delta\theta = 0$, and $\Delta d = 0$, and therefore, $D_{\theta d}(s, s') = 0$ (and vice versa). As $\Delta\theta$ or Δd increases, $D_{\theta d}(s, s')$ increases. If s and s' differ by $\Delta\theta = \theta_\delta$, and $\Delta d = d_\delta$, then $D_{\theta d}(s, s') = 1$.

Then we define a collinearity score $Col(s, s')$, which represents how collinear two given line segments s and s' are, as below:

$$Col(s, s') = \begin{cases} 1 - D_{\theta d}(s, s'), & \text{if } D_{\theta d}(s, s') \leq 1, \\ 0, & \text{otherwise.} \end{cases}$$

Figure 3.5. Examples of the scores $Col(s, s')$ for six sample input pairs

Thus, if the distance between s and s' is zero (or, s and s' are collinear), the collinearity score is equal to 1. If the distance is greater than 1, the collinearity score is zero. Figure 3.5 shows example scores of different line segment pairs. We regard s and s' as collinear for the merging step if $Col(s, s')$ is non-zero. The strictness of the collinearity test can be adjusted by the thresholds θ_δ and d_δ .

The merging step is done as follows. Let S be the entire set of detected segments from an image. For a given line segment s , let $S_{col}(s)$ be the set of all segments collinear with s :

$$S_{col}(s) = \{s_i \mid Col(s, s_i) \neq 0, \forall s_i \in S\}. \quad (3.2)$$

To check the overlap condition, we project all segments in S_{col} onto a virtual parallel line as illustrated in Figure 3.6. The two endpoints of each segment $s_i \in S_{col}$ can be represented as two parametric real numbers, $left(s_i)$ and $right(s_i)$ ($left(s_i) < right(s_i)$). The smaller parametric number is considered to be “lexicographically left.” We examine overlaps using the parametric numbers. When the left point of a segment s_j starts before the right point of a collinear segment s_i , or $left(s_j) \leq right(s_i)$, we merge s_i and s_j . The two end points of the

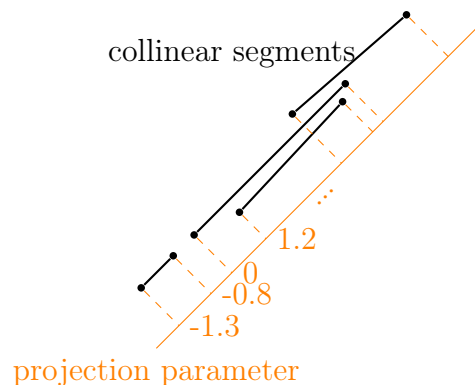


Figure 3.6. Projection to 1D parametric space.

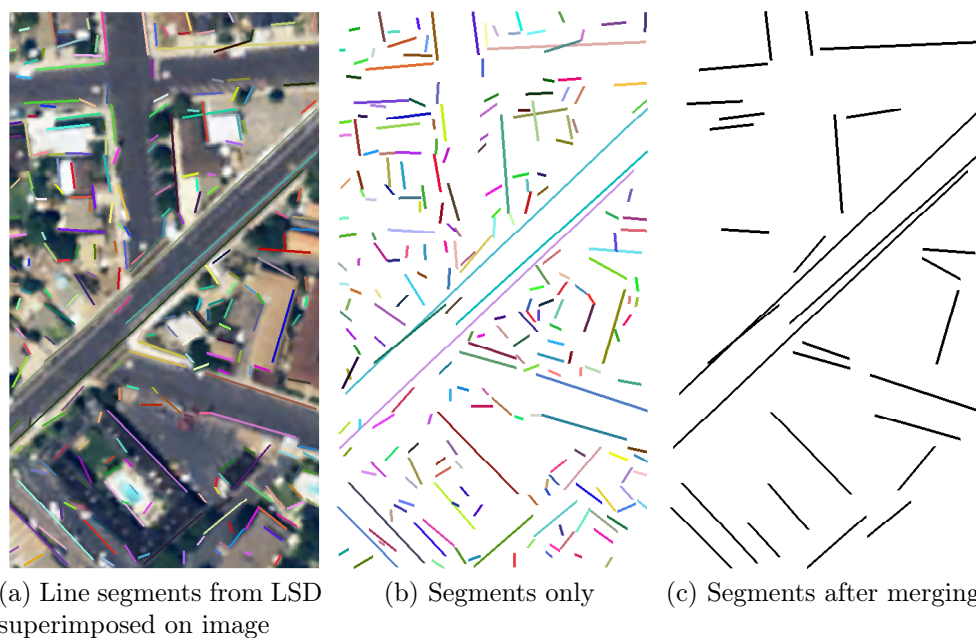


Figure 3.7. (a) and (b) show the detected lines using LSD. The coloring is just for visual help to distinguish individual line segments; (c) shows the 30 longest segments after the merging step.

new merged line are $\min(\text{left}(s_i), \text{left}(s_j))$ and $\max(\text{right}(s_i), \text{right}(s_j))$ (the lexicographically leftmost and rightmost points). Figure 3.7 illustrates the result of the merging step.

3.2.3 Transformation

We follow [Hartley and Zisserman \[2003\]](#) to solve the affine transformation between two images. Because an affine transformation has six degrees of freedom, one can solve an affine transformation from three control point pairs (CPPs) as follows. Given that a CPP of a point \mathbf{p} in image 1 and \mathbf{p}' in image 2 (\mathbf{p} and \mathbf{p}' are represented as 3×1 vectors in homogeneous coordinates), the 3×3 affine transformation $T : \mathbf{p} \rightarrow \mathbf{p}'$ satisfies:

$$\mathbf{p}' = T\mathbf{p}. \quad (3.3)$$

Then the affine transformation T that maps the points $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ to $\{\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3\}$ respectively is uniquely solved by:

$$P' = TP \quad (3.4)$$

or equivalently

$$T = P'P^{-1} \quad (3.5)$$

where

$$P := [\mathbf{p}_1 \mid \mathbf{p}_2 \mid \mathbf{p}_3] \quad (3.6)$$

$$P' := [\mathbf{p}'_1 \mid \mathbf{p}'_2 \mid \mathbf{p}'_3]. \quad (3.7)$$

Note that P, P' and T are all 3×3 matrices. In order for T to exist, P must be full rank, i.e., $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ should not be collinear.

We check all the possible transformations that can be acquired from the detected line segments' information, instead of estimating the transformation based on a feature description, because we assume that the two input images may include severe changes, in which case there is no appropriate point-level descriptor.

Let S and S' be the sets of detected line segments from image 1 and image 2, respectively. We choose any three non-parallel segments $\{s_i, s_j, s_k\} \in S$ and $\{s'_l, s'_m, s'_n\} \in S'$ from each set as candidates to check for correspondence. Both sets of three non-parallel segments (or their extended full lines) yield three intersection points, which we order counter-clockwise and denote $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ and $(\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3)$, respectively. In this setting, note that we do not know which point corresponds to which; there are three possible cases: the three points $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ may correspond to $(\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3)$, $(\mathbf{p}'_2, \mathbf{p}'_3, \mathbf{p}'_1)$ or $(\mathbf{p}'_3, \mathbf{p}'_1, \mathbf{p}'_2)$. Because we exclude the reflection transformations (aerial images are always collected above the ground), the counter-clockwise order itself should be preserved.

Denote a case C : $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)_C$ and $(\mathbf{p}'_1, \mathbf{p}'_2, \mathbf{p}'_3)_C$. Hypothesizing that these are true corresponding CPPs, we solve for the affine transformation T_C using equation 3.5. We repeat this process, testing every combination of three segments chosen from S and S' . We will discuss how to evaluate a matching score of $score(T_C)$ in section 3.2.4. The optimal transformation T^* is the one with the best matching score:

$$T^* = \arg \max_C score(T_C). \quad (3.8)$$

This process requires a huge number of calculations because the number of possible combinations is so large. In order to reduce computation, we limit the search to the most promising candidates by selecting the triples of segments only from the longest 30 line segments each in S and S' . Since the evaluation process is independent, parallel computation can be easily applied.

3.2.4 Matching Score

In this section, we define the matching score function $score(T_C)$ for a given hypothesis transformation T_C . Let S_{T_C} be the transformation of S by T_C . The idea is that if T_C is a true transformation, S_{T_C} and S' will be in harmony. In other words, many common segments in S_{T_C} and S' will be collinear.

For a segment $s \in S_{T_C}$, define a function $best(s, S')$ as:

$$best(s, S') = \max_{s' \in S'}(Col(s, s')). \quad (3.9)$$

This function searches for the segment $s' \in S'$ that is most collinear with s . Then we define a matching score function:

$$score_{S \rightarrow S'}(T_C) = \frac{1}{|S_{T_C}|} \sum_{s \in S_{T_C}} (best(s, S')), \quad (3.10)$$

where $|S_{T_C}|$ is the cardinality of S_{T_C} .

Note that S can be transformed into the coordinate system of S' by T_C , and S' can be transformed into the coordinate system of S by T_C^{-1} . Since the mapping score is not symmetric, we calculate the mapping score for the latter direction as well:

$$score_{S' \rightarrow S}(T_C^{-1}) = \frac{1}{|S'_{T_C^{-1}}|} \sum_{s' \in S'_{T_C^{-1}}} (best(s', S)). \quad (3.11)$$

The final mapping score is their average:

$$score(T_C) = \frac{1}{2}score_{S \rightarrow S'}(T_C) + \frac{1}{2}score_{S' \rightarrow S}(T_C^{-1}). \quad (3.12)$$

Figure 3.8 visualizes the choice of the three line segments (per image) that yields the transformation with the best matching score. It also shows the three intersections in each image, for which there are three possible ways to pair them. The color of these points (red, green, and blue) represents the pairing relationship. Figure 3.9 shows the registration result and how S and S' are finally matched. In Figure 3.10, we compare our result with Fedorov et al. [2003a]'s work [Fedorov et al., 2003b] and image registration using SIFT (open source [Vedaldi and Fulkerson, 2010]), two algorithms whose implementations we can access.



(a) A choice of three line segments from Image 1, and their intersections.



(b) A choice of three line segment choice from Image 2, and their intersections.

Figure 3.8. The three line segments highlighted in red in the two images is the choice that yields the optimal transformation T^* . The three circles in each image indicate the intersection points of the three chosen segments. The circle color represents correspondence between the points.



Figure 3.9. (a) shows the registration result using the optimal transformation T^* . (b) visualizes how S and S' are finally matched.

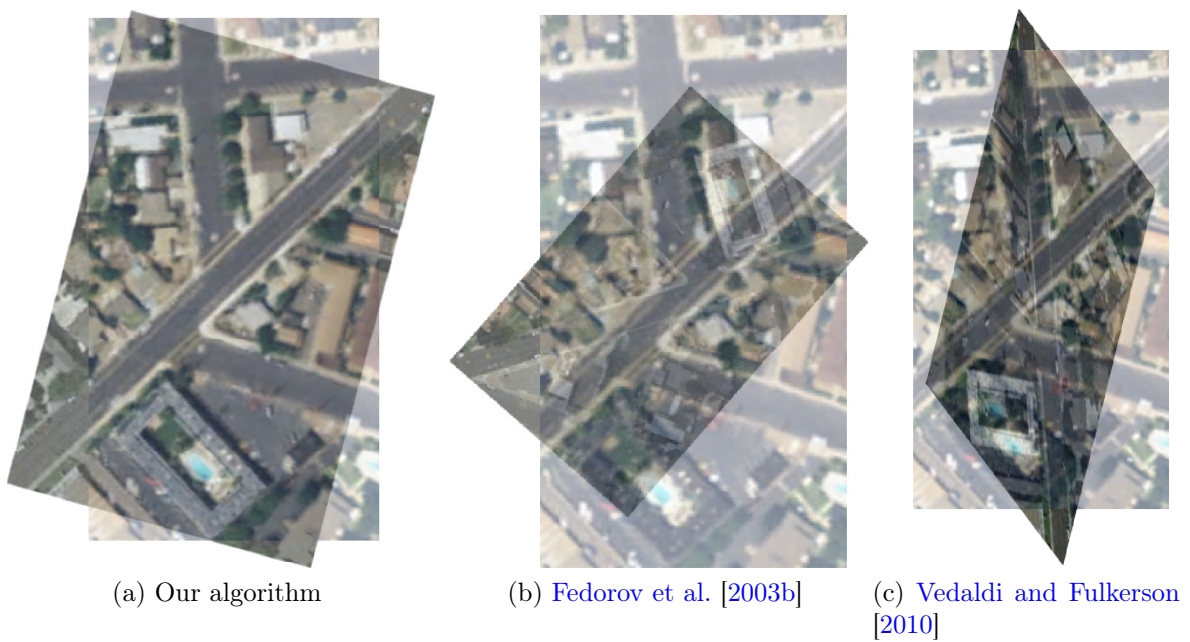


Figure 3.10. Result comparison: (a) Our algorithm (b) Fedorov et al. [2003b] (c) SIFT implemented by Vedaldi and Fulkerson [2010]

3.2.5 Filtering

For the purpose of reducing unnecessary computation, before measuring a score of the hypothesis transformation T_C , we examine T_C by decomposing it into the six elements of the affine transformation: t_x (translation in x), t_y (translation in y), s_x (scale in x), s_y (scale in y), θ (rotation angle), k (shear). The “true” transformation should have these elements in certain limited ranges. For example, a t_x or t_y so large that it means that the two input images are not even overlapped is not appropriate. If T_C has a very large s_x or s_y , it is highly likely to be a wrong answer. The rules we use are as follows, considering T_C to be valid only when:

- translation: $|t_x|, |t_y| < \min(\text{image height}, \text{image width})$;
- scale: $1/3 < s_x, s_y < 3$;
- skew: $|k| < 0.2$; and
- rotation: no constraint.

If T_C violates one of these conditions, we do not need to calculate its matching score. We discard the choice and keep repeating the process with another choice. If we have stricter rules, we can save more computations by discarding more frequently.

3.3 Results

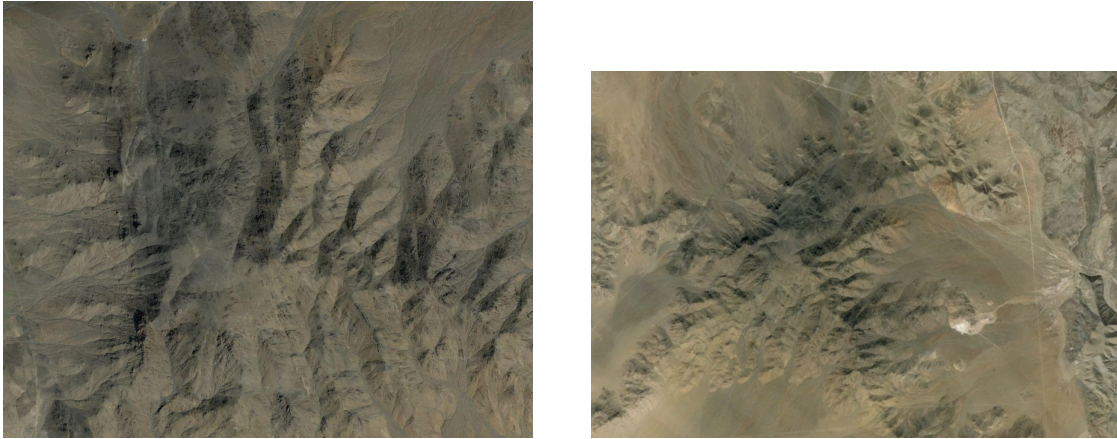
We collect image pairs from various sources, and demonstrate them in Figure 3.11. From Figure 3.12 to Figure 3.18, we show our results compared to Fedorov et al. [2003b] and Vedaldi and Fulkerson [2010]. We order the input image pairs in registration difficulty order, from the easiest to hardest. As the registration difficulty increases, other algorithms start to fail. We show that our registration algorithm is outstanding, especially for the input images of high registration difficulty.



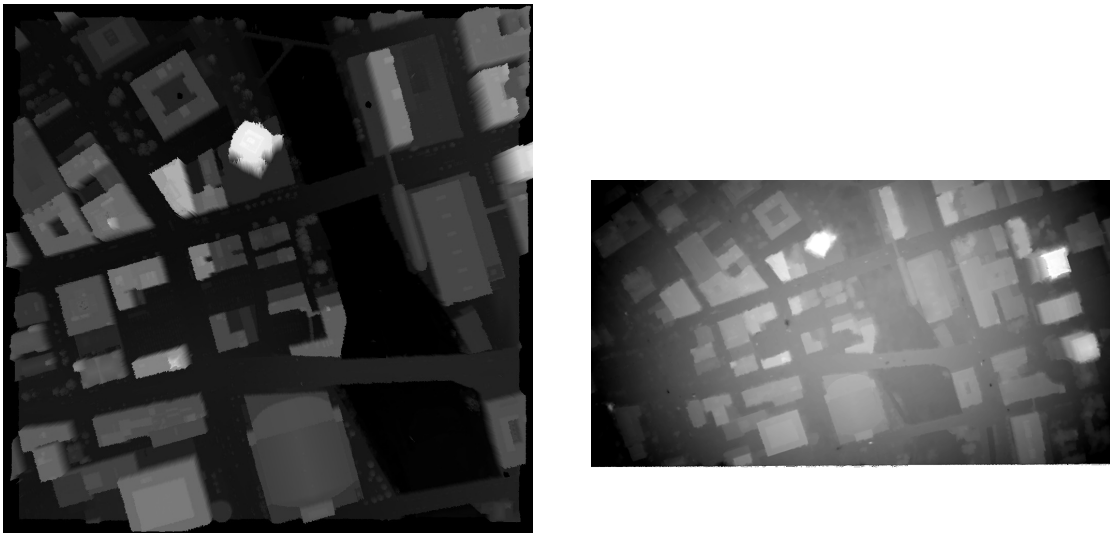
(a) greentest: Different frames from an aerial video. Courtesy of Lawrence Livermore National Laboratory (LLNL).



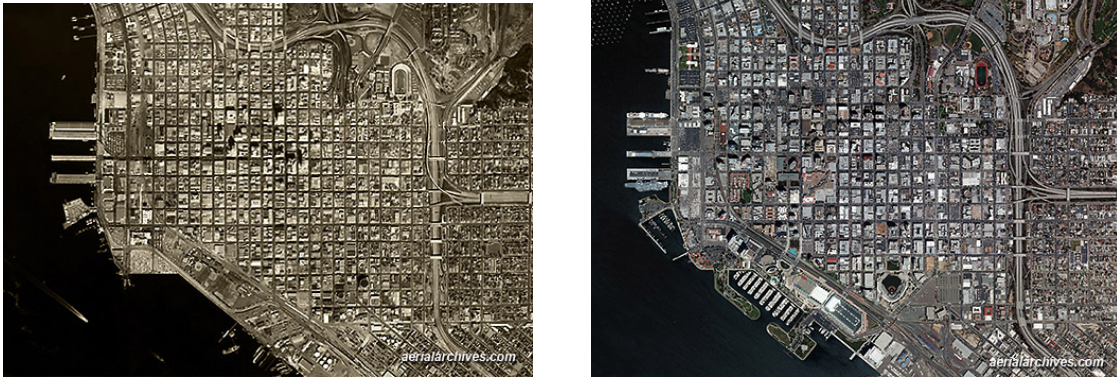
(b) circular_road: Different frames from an aerial video. Courtesy of LLNL.



(c) mountain: Google Satellite (left) and Google Earth (right).



(d) depth_map: A LiDAR depth (left) and an estimated depth (right). Reproduced from [Kim et al. \[2014a\]](#).



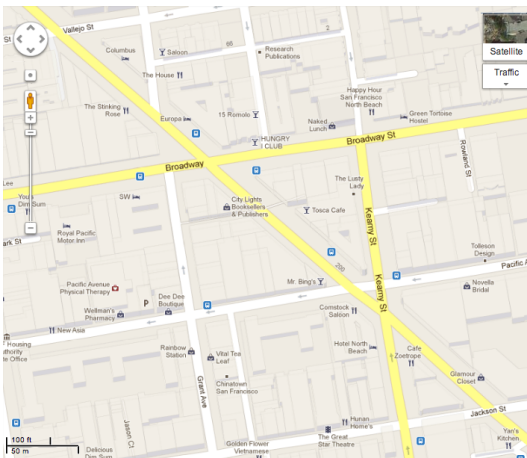
(e) coast: Unknown somewhere. Excerpt from Google image search.



(f) livermore1: A random location in Livermore in different years. Excerpt from Google Earth.



(g) livermore2: A random location in Livermore in 2004 and 2012. Excerpt from Google Earth.



(h) map: Google map (left) and Google Satellite (right) of a random location in San Francisco

Figure 3.11. Our input dataset.

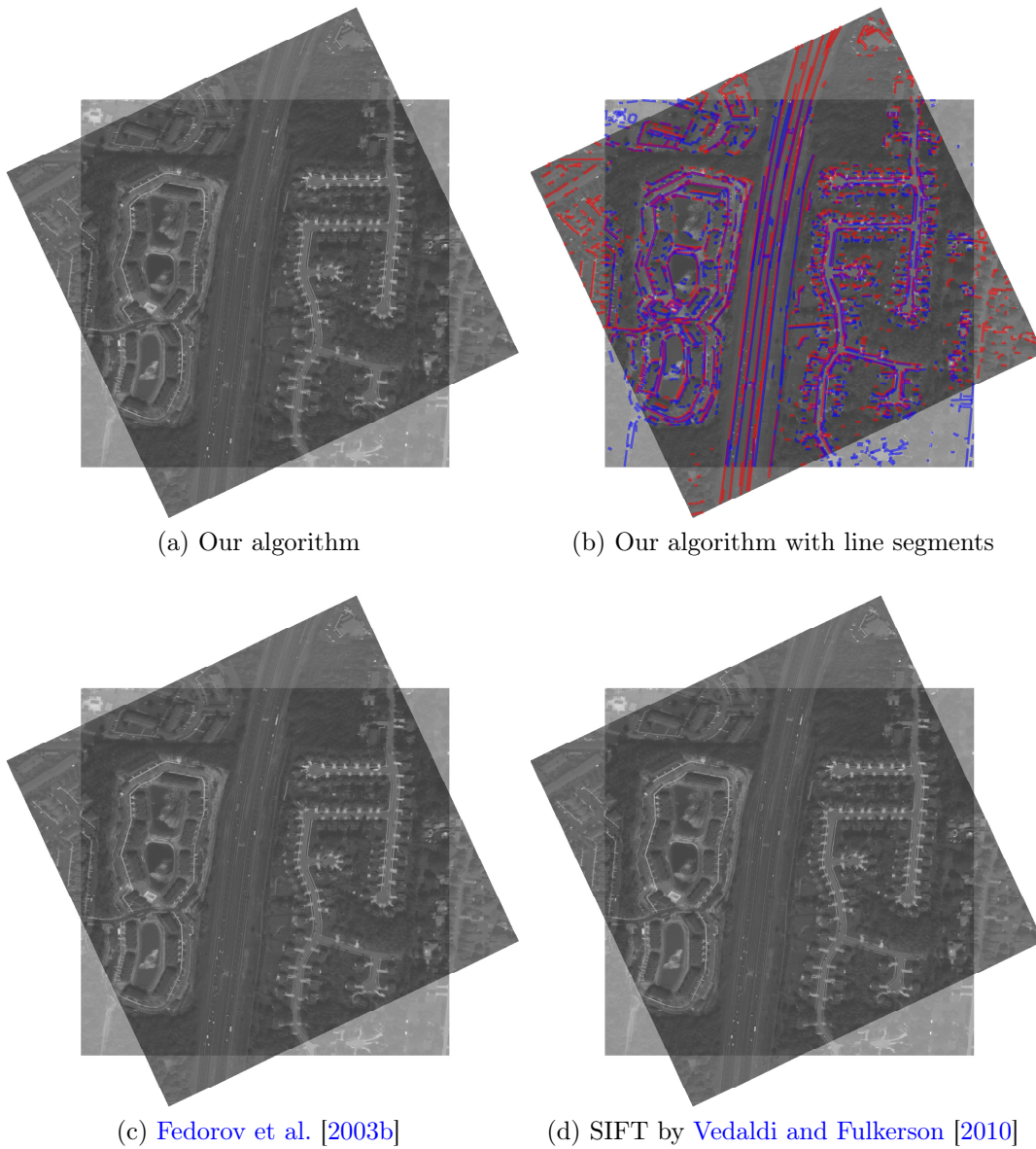


Figure 3.12. Result: greentest

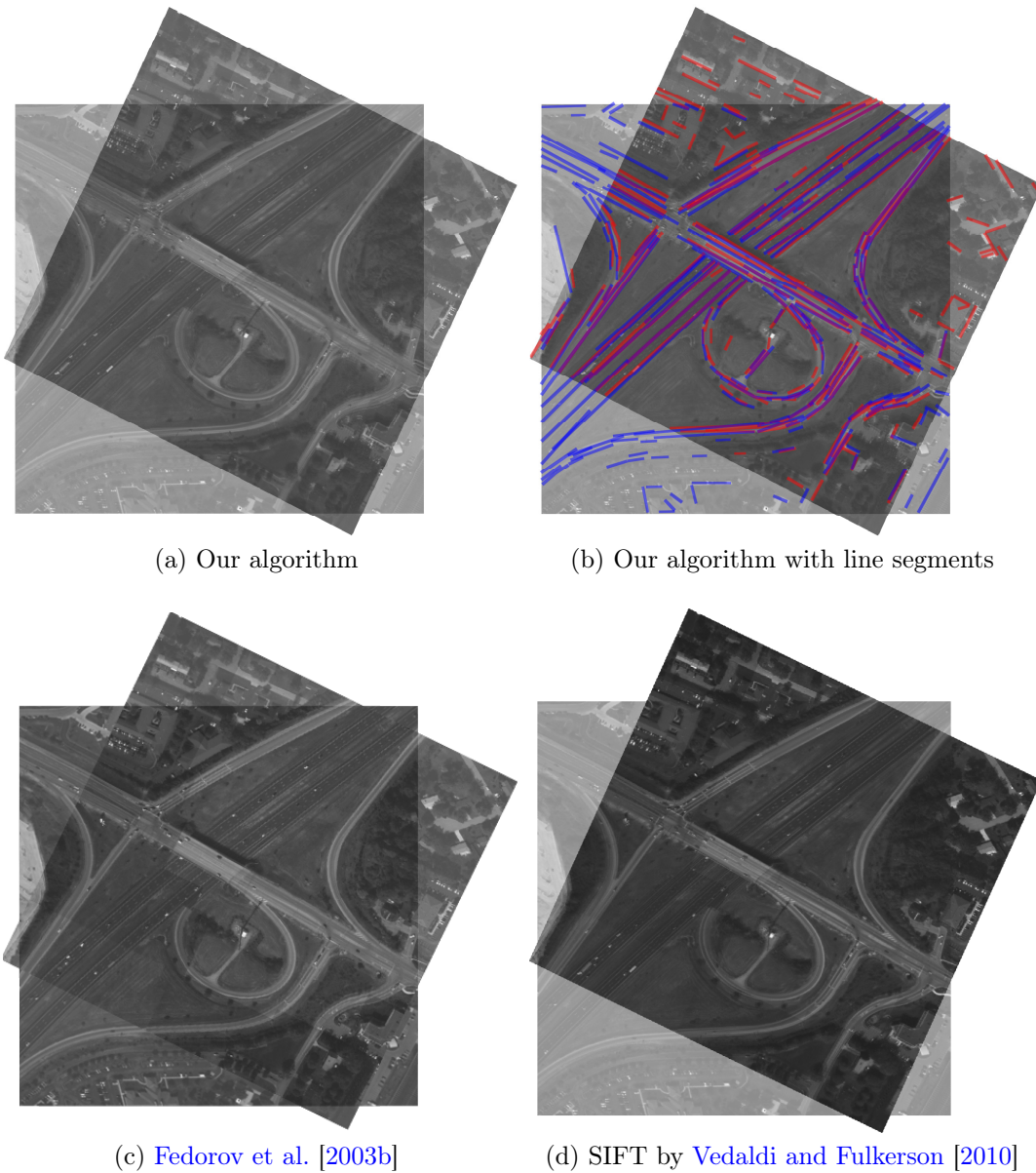
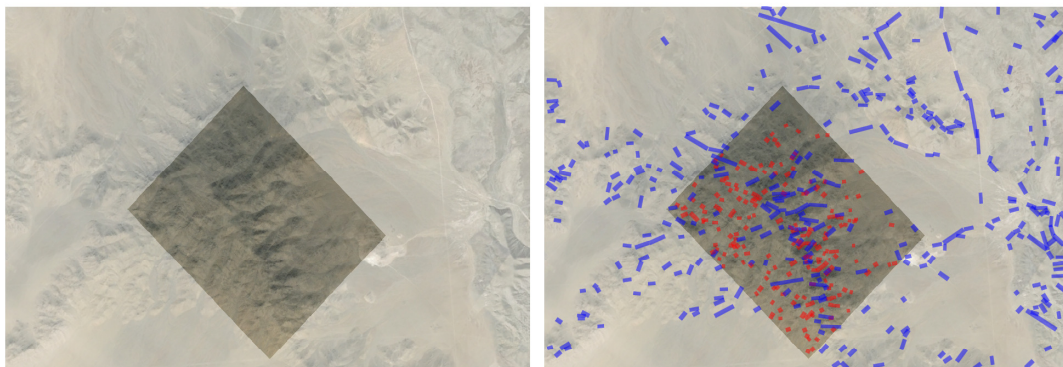


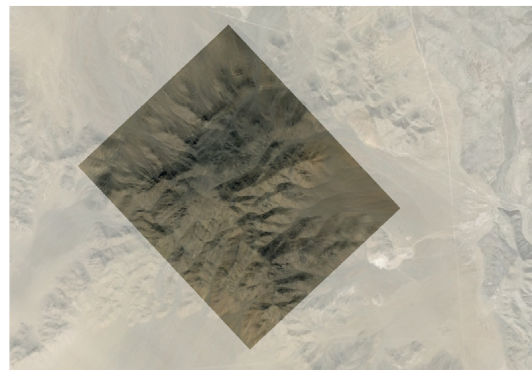
Figure 3.13. Result: circular_road



(a) Our algorithm

(b) Our algorithm with line segments

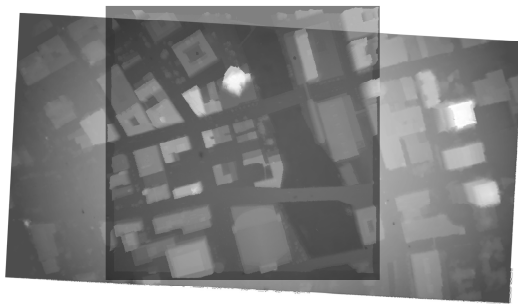
Registration Failure



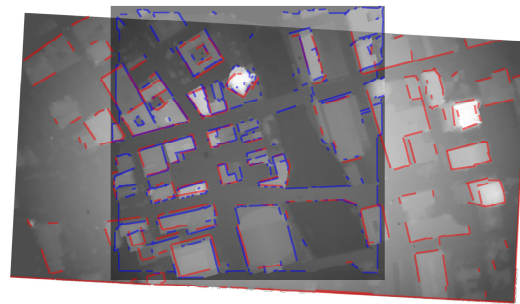
(c) Fedorov et al. [2003b]

(d) SIFT by Vedaldi and Fulkerson [2010]

Figure 3.14. Result: mountain

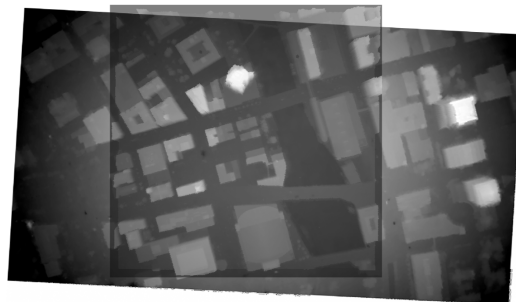


(a) Our algorithm



(b) Our algorithm with line segments

Registration Failure



(c) Fedorov et al. [2003b]

(d) SIFT by Vedaldi and Fulkerson [2010]

Figure 3.15. Result: depth

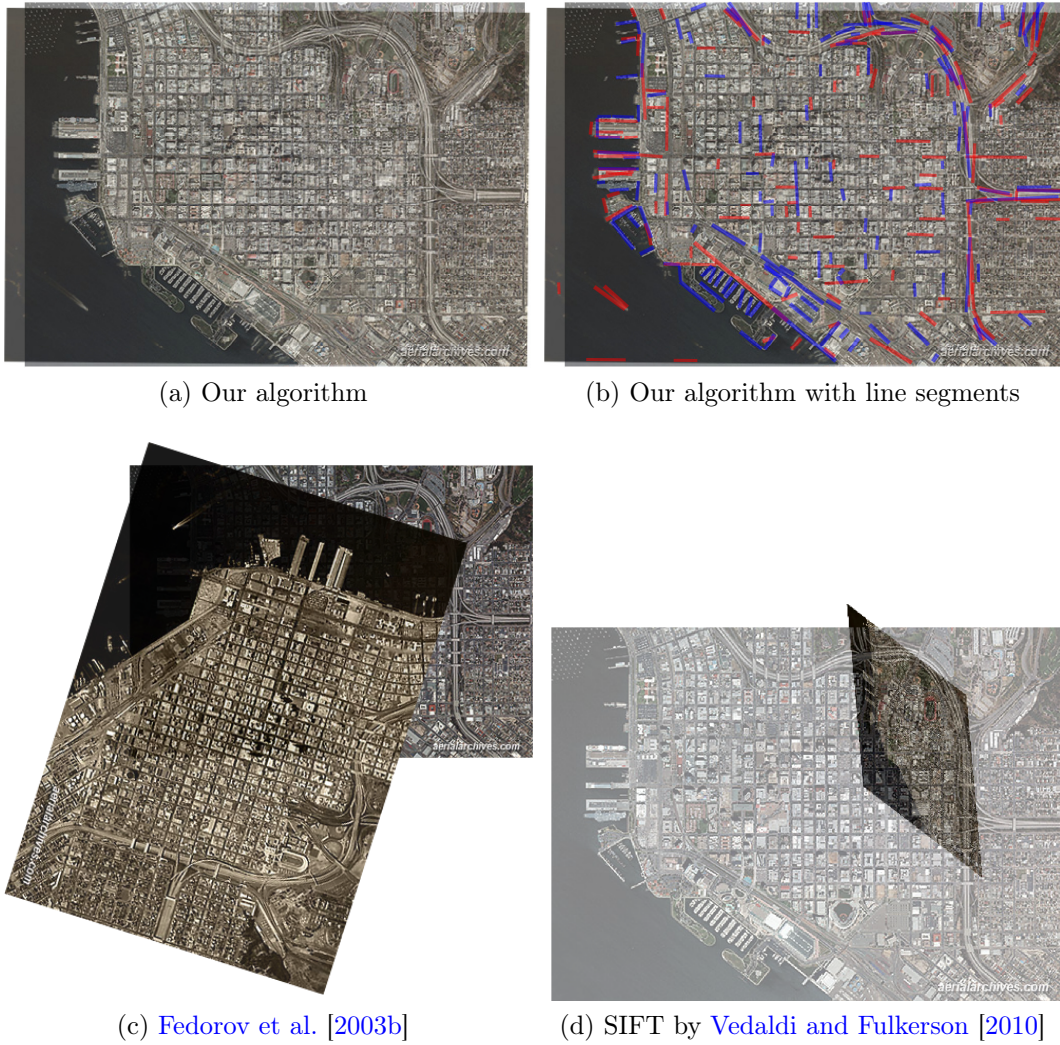
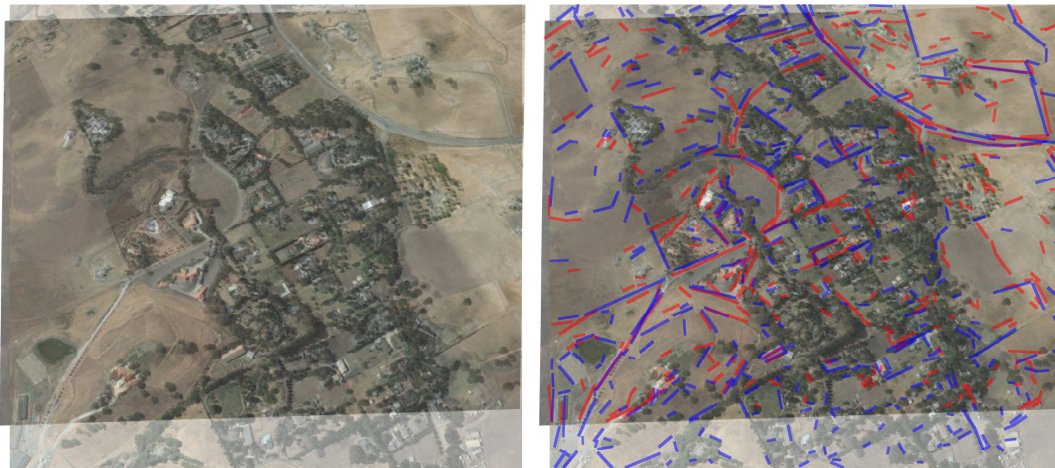


Figure 3.16. Result: coast



(a) Our algorithm

(b) Our algorithm with line segments

Registration Failure

Registration Failure

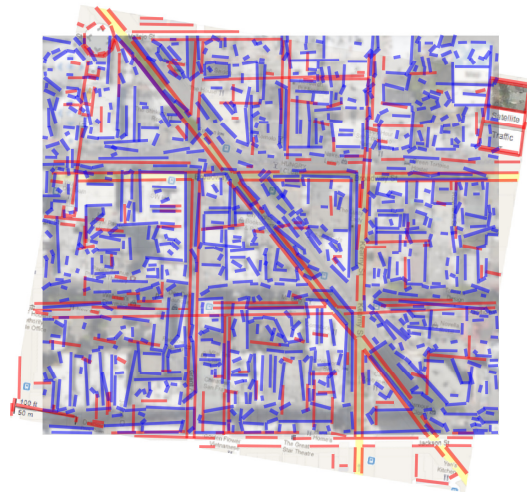
(c) [Fedorov et al. \[2003b\]](#)

(d) SIFT by [Vedaldi and Fulkerson \[2010\]](#)

Figure 3.17. Result: livermore1



(a) Our algorithm



(b) Our algorithm with line segments

Registration Failure

Registration Failure

(c) Fedorov et al. [2003b]

(d) SIFT by Vedaldi and Fulkerson [2010]

Figure 3.18. Result: map

3.4 Summary

We propose a new segment-based registration system for disparate images with a wide range of registration difficulties. In the first step of our registration process, we detect line segments in each image using the LSD algorithm. Next we conduct the merging step on the detected line segments. Finally, using the merged line segments as input, we generate possible hypothesis affine transformations by choosing three segments in each image. After scoring each hypothesis transformation based on the score metric, the highest-scoring one is selected.

This algorithm can easily be extended to projective transformations or higher degrees of transformations by choosing four or more segments instead of three, and then solving a hypothesis transformation based on the choice. This is exactly the same procedure; however, it will have a larger searching space and more intense computation.

Chapter 4

Linear Feature Matching With DUDE Descriptor

4.1 Introduction

In the previous chapter, we showed that the distribution of line segments can be an important tool to enable image registration in cases of dramatically inconsistent appearance. However, since examining huge numbers of combinations is computationally very expensive, we discuss how we can create a computationally efficient descriptor for line segments. We propose a novel descriptor system named DUDE (DUality DEscriptor) that uses a histogram based on a weighted transform of lines to an r - θ dual space (or parameter space) of lines. By exploiting line-point duality, DUDE captures geometric relationships very efficiently. Our experimental results show that DUDE shows equivalent or better performance to that of the state-of-the-art with significantly less computation cost.

This chapter includes three main factors. First, to acquire repeatable and consistent line segments across disparate images, we employ a method called ensemble of randomized segmentation originally proposed for image segmentation [Kim et al., 2014b] to generate multiple line configurations. Its randomized merging process allows us to collect as many consistent lines as possible between images. Second, by exploiting a dual space, we provide a computationally efficient methodology to capture the relative geometric distributions of line segments in an image. Third, we introduce two additional methods to filter matches that are likely incorrect: exploiting intentional random perturbation of line segments and checking a similarity measure of two sets of line segments. These processes enable us to identify more stable and robust matches.

4.2 Proposed Method

Before we describe our proposed feature detector and descriptor in detail, we remind readers that the definitions and background about feature detection and description appear in

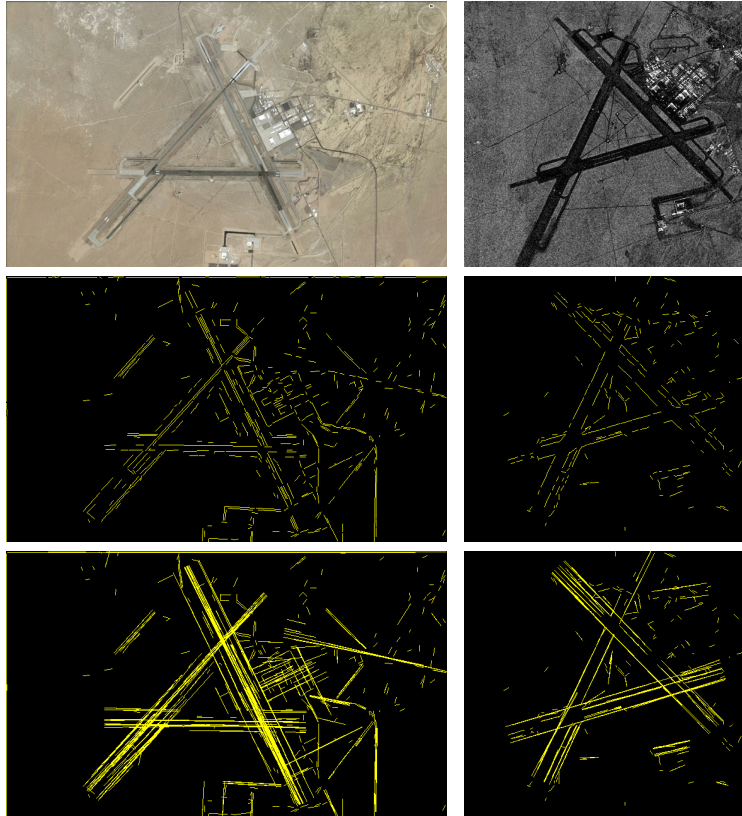


Figure 4.1. An example of two disparate images and their line segments: An EO (Electro-Optical) and a SAR (Synthetic Aperture Radar) image respectively from [Sandia National Laboratories \[2016\]](#) (top), initial line segments from LSD [[Grompone von Gioi et al., 2012](#)] (middle), and the proposed randomly merged line segments (bottom).

Section 2.1.

4.2.1 Feature detection

Similar to other well-known features (e.g., Difference of Gaussian, SIFT), we denote each feature as $[x_i, y_i, s_i, \theta_i]$ for its location, scale and orientation, respectively. Our primitive idea is to derive one feature from one line segment so that (x_i, y_i) is its midpoint, s_i is a half of the length, and θ_i is the angle of the line segment. One could simply use the initial line segments detected; however, in the case of disparate imagery, most initial line segments are inconsistent across images (i.e., low repeatability), due to dramatic appearance changes. Consequently, corresponding line segments yield different locations and scales, as shown in Figure 4.1 (middle row).

To guarantee consistent feature extraction for disparate imagery, we propose generating multiple *segmentations* of line segments from the initial line segments by randomly merging

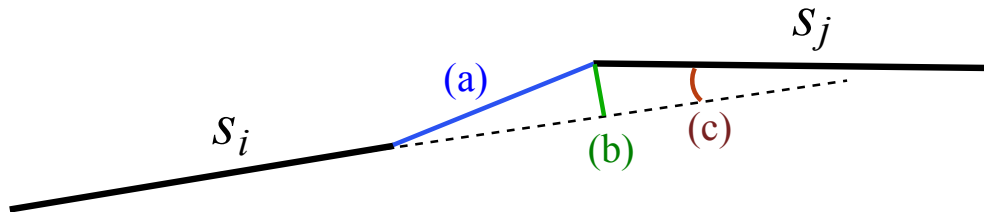


Figure 4.2. Merging criterion for the randomized merging process; [a] shortest distance (in pixel), [b] perpendicular distance (in pixel), and [c] angle (in degree) between line segment i and j .

the lines, motivated by the ensemble creation strategy through randomization [Kim et al., 2014b], a strategy designed for image segmentation. Before describing how we adapt this approach to our problem, let’s discuss image segmentation briefly. It is quite common that state-of-the-art image segmentation methods begin with a set of superpixels [Achanta et al., 2012] (perceptually meaningful atomic regions, well-aligned with edges). Image segmentation can be thought of as grouping image superpixels; many different metrics to calculate similarity between two adjacent superpixels and different schemes to cluster superpixels based on such metrics has been proposed. The key idea of Kim et al. [2014b] is to merge superpixels step by step in a hierarchical manner, and re-calculate similarities at each hierarchical level until convergence. They showed that such hierarchical segmentation randomized the order of merging and generated different image segmentation results. Interestingly, this process can be applied to our line segment merging process, albeit in a very different application. In our case, “line segments” are compared to “superpixels” and generating multiple “merged line segments” can be compared to generating multiple “image segments.” We build multiple bottom-up hierarchical merging processes from the initial line segments ($\#$ hierarchies = 10 in our case). These processes allows us to explore as many potential line merging configurations as possible. The bottom images in Figure 4.1 show that such merging processes produce more corresponding consistent features across images, increasing repeatability rates, compared to an initial line segment detection result.

The line merging process is as follows. We again extract a set of initial line segments from an image using the Line Segment Detector (LSD) algorithm [Grompone von Gioi et al., 2012]. Given the initial line segments, we build a graph where each *node* represents a line segment (as if superpixel in the image segmentation application), and each *edge* connects two neighboring line segments with a corresponding weight (merging criterion). Then we incrementally merge line segments as in Kim et al. [2014b]. While constructing a hierarchy, we update the graph and compute new edge weights. The merging criterion between line segment i and j consists of three terms: shortest distance $\delta_{1,i,j}$, perpendicular distance $\delta_{2,i,j}$, and angle $\delta_{3,i,j}$ between the two line segments (Fig. 4.2). The edge weight $w_{i,j}$ is computed as

$$w_{i,j} = \max(0, (1 - \delta_{1,i,j}/\alpha)) \cdot \max(0, (1 - \delta_{2,i,j}/\beta)) \cdot \max(0, (1 - \delta_{3,i,j}/\gamma)) \quad (4.1)$$

where α , β and γ are predefined thresholds (we used 80, 16, and 15°). Once all hierarchies are constructed, all merged line segments are integrated and any duplicates (two merged lines that are almost the same) are removed. The merged line segments from the hierarchies are then used for extracting line features. Note that we discussed a different method of merging lines in Equation (3.1) that relied on midpoints, and thus merged overlapped lines. In this section, since we aim to produce more various lines, we modified the merging criterion to use the edge weights in Equation (4.1), which is more flexible than the former (Equation (3.1)): for example, consider two line segments that are not overlapped, but collinear. With the former (Equation (3.1)), regardless of how far they are from each other, as long as the two line segments are collinear, they are merged. However, with the latter (Equation (4.1)), α can control how much the two line segments are targets of merging based on their shortest distance $\delta_{1,i,j}$. Finally, we create features from the merged line segments. Since we generate a number of variously merged line segments, our extracted features include various midpoints and scales. Specifically, for each (final) line segment, we take its midpoint as a feature location (x, y) , and its radius as feature scale (s) and its orientation as feature orientation (θ). We will denote our feature detection described above as MMID (Merged-Midpoints).

4.2.2 Feature description

The main idea of the DUDE descriptor is to take advantage of line-point duality by transforming lines into points in dual space. An infinite line l in 2D has two degrees of freedom, which can be represented by a number of different duality transforms. We use r - θ dual space, which encodes the two degrees of freedom as the normal angle θ measured from the coordinate frame's x-axis and the orthogonal distance r from the origin (Figure 4.3a). We denote this:

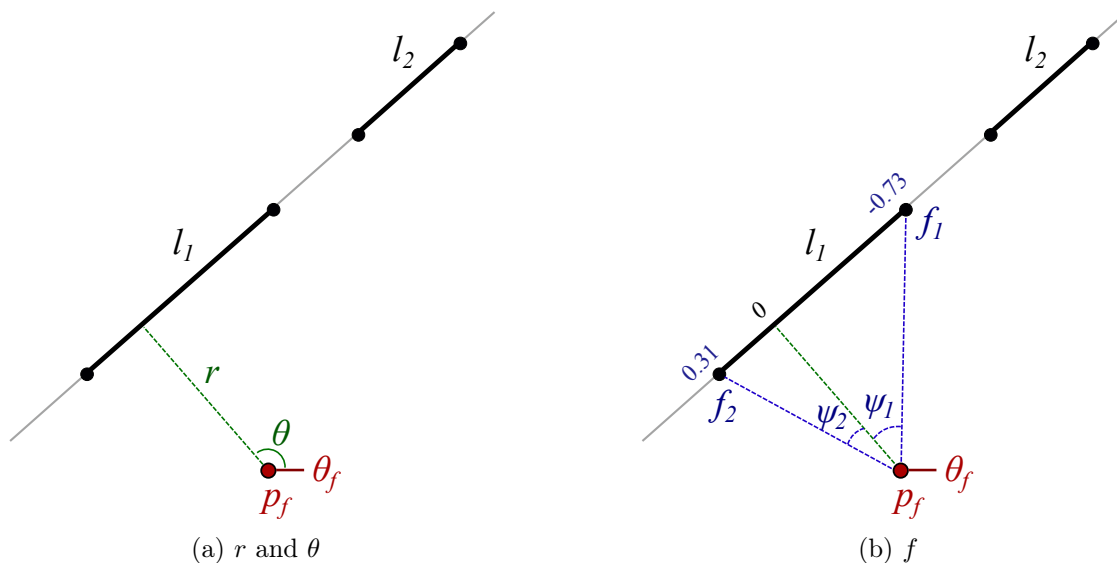
$$l \rightarrow (r, \theta) \text{ where } x \cos \theta + y \sin \theta - r = 0. \quad (4.2)$$

Such a line can be uniquely represented one of two ways, as either $(r \in \mathbb{R}, \theta \in [0, \pi))$ or $(r \in \mathbb{R} \geq 0, \theta \in [0, 2\pi))$, where \mathbb{R} represents the set of real numbers. We follow the latter convention.

Because the parameters, r and θ , are for an infinite line, collinear line segments s_1 and s_2 share the same r and θ . In order to encode where a line *segment* occurs on an infinite line l , we introduce another variable f , which represents how far an endpoint is from the orthogonal projection onto l from a feature point p_f (Figure 4.3b). We calculate f as $\cos(\psi)$, where ψ is the angle between the orthogonal vector to l from p_f and the vector from p_f to the endpoint. Naturally, f has a value in $[-1, +1]$.

Now we can denote a line segment as $[r, \theta, f_1, f_2]$ where $f_1 < f_2$, instead of $[x_1, y_1, x_2, y_2]$. This representation emphasizes co-linearity. In Figure 4.3b, two line segments l_1 and l_2 share the same r and θ , but different f values.

Our descriptor design is as follows. For a given feature $[x_i, y_i, s_i, \theta_i]$, we first identify a set of line segments \mathbf{S}_i that are within a circle whose center is (x_i, y_i) , and radius qs_i ,

Figure 4.3. Dual representation r, θ, f_1, f_2 of a line segment

where q is a parameter for local range of interest. (We regard a line segment as *within* a circle if any part of the line segment lies within the circle.) Then we define a 3D histogram. The first dimension for binning relates to r . We uniformly divide $[0, qs_i]$ into n_r bins. The second dimension for binning relates to θ ; we uniformly divide $[0, 2\pi)$ into n_θ bins. The third dimension for binning, which relates to f_1 and f_2 , is somewhat different. As shown in Figure 4.4, we divide the range from 0 to 1 (and also to -1 symmetrically) into bins using a log scale. Because f_1 and f_2 denote the range of each line segment, segments are binned as a *range*, contributing to bins by the coverage percentage. In Figure 4.4, we provide an example of the range histogram when $n_f = 6$, $f_1 = -0.4$ and $f_2 = 0.1$. In this manner, each $[r, \theta, f_1, f_2]$ of $s \in \mathbf{S}_i$ is accumulated in the 3D histogram. By concatenating the (r, θ, f) dimensions of the 3D histogram to form a 1D vector, we have a $(n_r \times n_\theta \times n_f)$ -dimension descriptor. We set $q = 10$, $n_r = 5$, $n_\theta = 5$, and $n_f = 10$ for our experiments.

Figure 4.5 illustrates the summary of the whole process. For a given feature $(x, y, r$ and $\theta)$ are shown as the center of the blue circle, its diameter, and its line indicator, respectively (top left). Red line segments are the line segments in the image within the feature region. For each line segment s_j , we calculate its $(r_j, \theta_j, f_j^1, f_j^2)$ and identify relevant (r, θ) -bins (bottom left) and f bins (top right) cylindrical bins. DUDE descriptors can be thought of as 3D cylindrical bins (bottom right). For visualization purposes, we superimpose the f -bins *inside* the 2D (r, θ) -bins as in Figure 4.6. Note that, for a segment, the 2D visualization approximately indicates where the (infinite) line containing that segment is (r, θ) -bin and how long the segment is (f) -bins. Using this visualization scheme, we illustrate three example descriptors (at manually chosen feature points) for a disparate image pair in Figure 4.7, which shows DUDE descriptors are consistent across corresponding features, and at the same time,

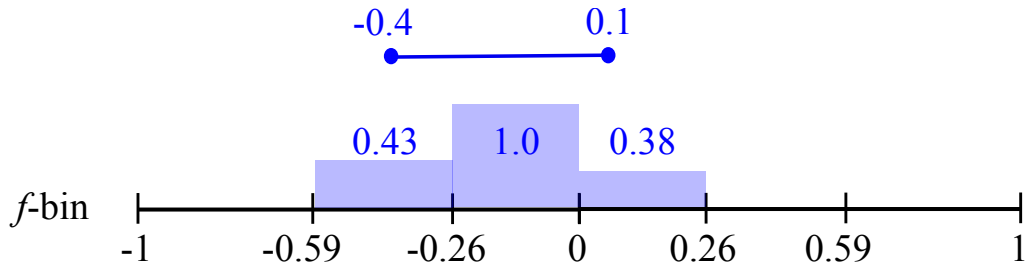


Figure 4.4. An example of f -binning for a line segment with endpoints $f_1 = -0.4, f_2 = 0.1$ and six bins.

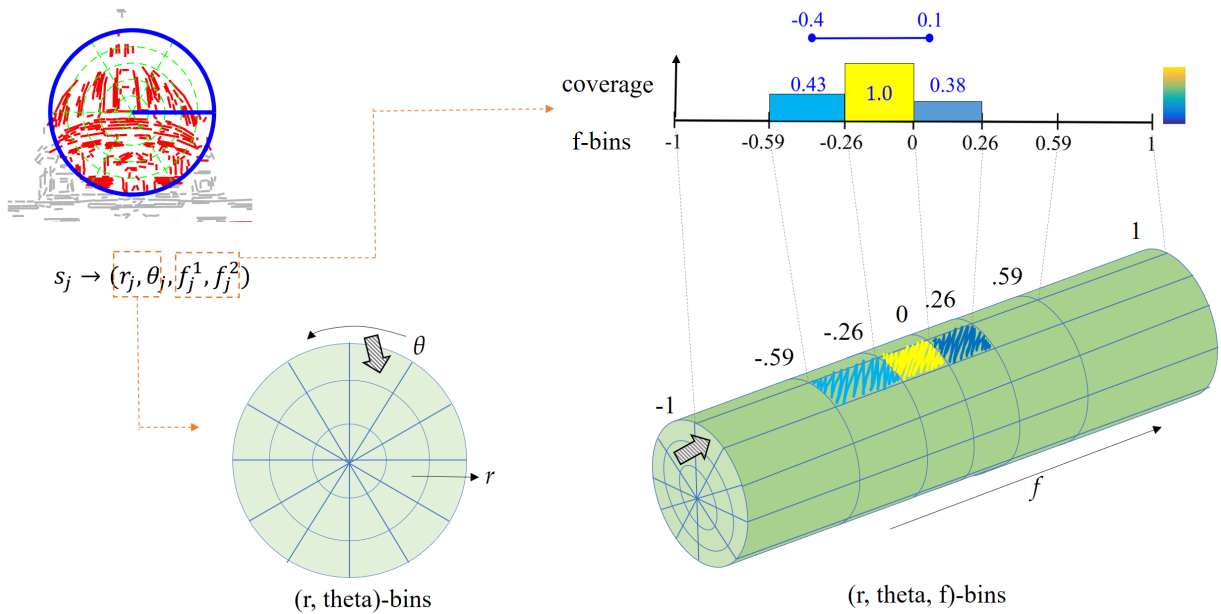


Figure 4.5. The summary of DUDE descriptor process.

distinguishable within an image.

There are two underlying difficulties for any line-based approach to overcome. First, there is the case that one long line segment in an image is detected as multiple short segments in the counterpart image. We lessen this disconnected detection problem by the nature of our descriptor design. Because collinear line segments share the same r and θ , and their f ranges are accumulated, the disconnection does not cause much difference in descriptors. Secondly, slight changes of endpoints can cause changes in r and θ values. We solve this problem by intentional *perturbation* of endpoints. We duplicate each segment d times, while randomly perturbing the endpoints of the additional segments within ± 3 pixels, both in

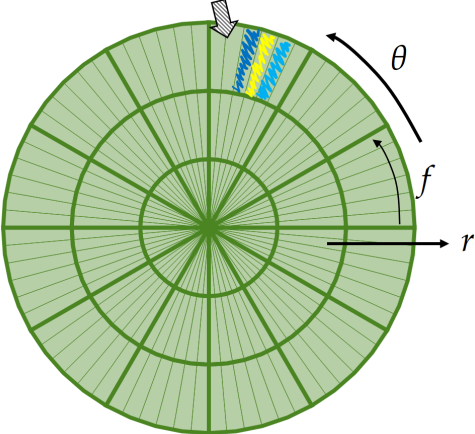


Figure 4.6. 2D visualization of DUDE

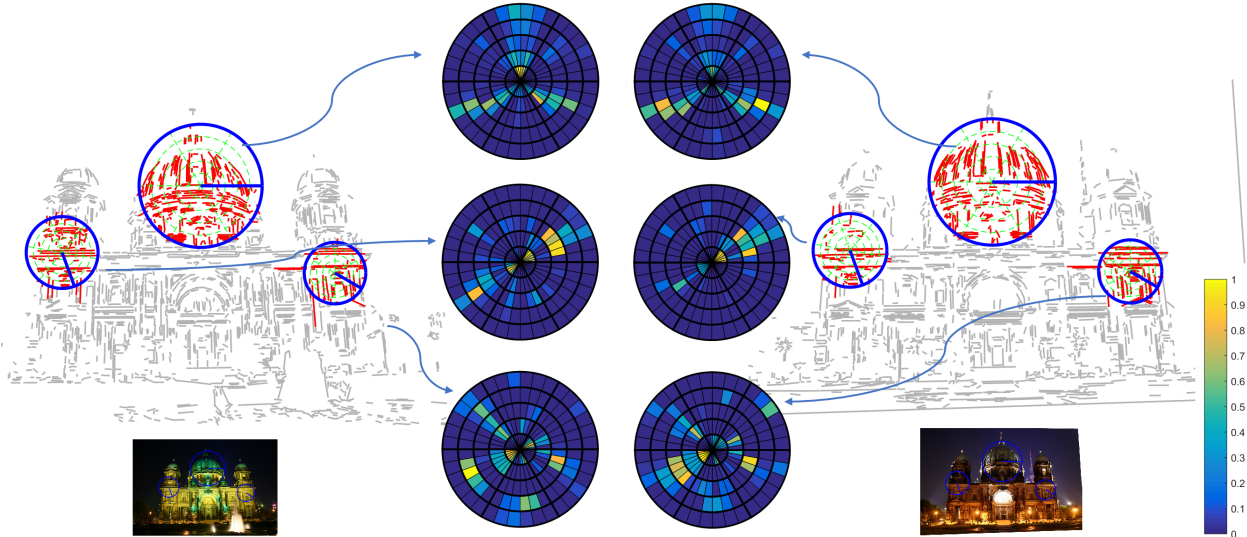


Figure 4.7. Three example DUDE descriptors



Figure 4.8. Line segment perturbation. In addition to the originally detected line segments, we intentionally duplicate each one d times while randomly perturbing its endpoints within ± 3 pixels. By doing so, we blur the DUDE descriptors and make them less sensitive to unstable endpoint detection.

x and y . Figure 4.8 shows an example of perturbed line segments ($d = 2$). This can be regarded as blurring the histograms and making them less sensitive to unstable endpoint detection.

4.2.3 Feature Matching

Matching consists of pairing every $d_i \in D_1$ with the most similar $d_{1NN(i)} \in D_2$ (in other words, the nearest neighbor (1NN) in D_2). We use the χ^2 histogram similarity metric for descriptor similarity measure, which is more suitable to our histogram design than the Euclidean distance (L_2). (See Appendix A for the definitions of the χ^2 metric and other possible histogram similarity metrics.)

We conduct two additional processes to identify trustworthy matches. First, we can actively exploit the randomness (from our intentional perturbation) not only in encoding descriptors but also in matching descriptors. Thanks to the randomness in our descriptors, we may acquire different matching pairs every time. We hypothesize that those matches which are preserved over several trials are more stable and highly likely correct. Therefore, we conduct the descriptor encoding and matching steps k times, and remove unstable matches. The intentional perturbation makes descriptors not only less sensitive to endpoint detection, but also provides information about which matching pairs are more likely correct.

As an optional further enhancement, one can thoroughly check each match by examining two sets of line segments related to the matched features. In other words, because the matches are acquired only by the similarity of descriptors, one can confirm the match after directly comparing the two set of line segments related to the matched features.

For this purpose, we define a similarity metric for two sets of line segments, $\mathbb{C}(S_1, S_2)$. First, for $s_1 \in S_1$ and $s_2 \in S_2$, we define a “coverage” metric, $c(s_1, s_2)$: we uniformly sample points on s_1 and check, for each sample point, if the (closest) distance from it to s_2 is less

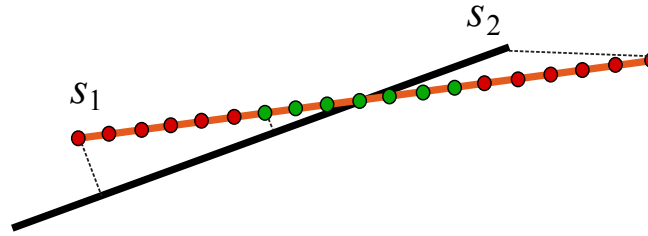


Figure 4.9. Line segment coverage calculation. We sample points on s_1 and calculate the closest distance from each sample point to s_2 , and check if the distance is within a threshold. The coverage score between two segments, $c(s_1, s_2)$, is the fraction of the number of “covered” sample points (green) to the total number of sample points.

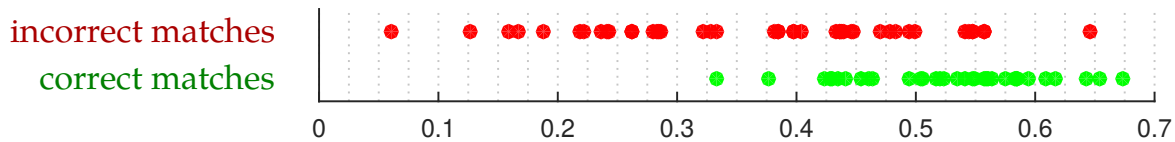


Figure 4.10. Threshold for our coverage metric $\mathbb{C}(S_i, S_j)$ (x -axis). Correct matches are shown in green, and incorrect matches are shown in red. Generally if $\mathbb{C}(S_i, S_j)$ is less than around 0.4, the match is highly likely incorrect.

than a certain threshold (Figure 4.9); $c(s_1, s_2)$ is the fraction of the number of “within” sample points to the total number of sample points. Note that this metric is not symmetric.

Then, we define overlap of a segment s_1 by a set S_2 :

$$C(s_1, S_2) = \sum_{s_2 \in S_2} c(s_1, s_2). \quad (4.3)$$

We say s_1 is *covered* by S_2 if $C(s_1, S_2) \geq \tau$ (we use $\tau = 0.5$). Finally, we define similarity of a set S_1 to a set S_2 , $\mathbb{C}(S_1, S_2)$ as the fraction of covered segments in S_1 by S_2 . Because this metric is not symmetric, we calculate the average of $\mathbb{C}(S_1, S_2)$ and $\mathbb{C}(S_2, S_1)$.

Using a randomly chosen image pair input, for a set of matches, we plot the distribution of the similarity, and visualize correct matches with green and incorrect matches with red in Figure 4.10. We found that 0.3 – 0.5 is a reasonable range. We remove matches whose scores are less than a threshold $\rho = 0.4$.

There are a few things to note. First, because we regard this second enhancement by the coverage computation as optional, in the evaluation section, we measure performance with and without it: DUDE denotes the matching results without it, and DUDE-F (filtered) denotes the matching results with it. Second, even though one could do this examination for the $n_1 \times n_2$ feature combinations from the beginning without any kind of descriptors, with the help of DUDE descriptors, we only need to conduct the test less than or equal to

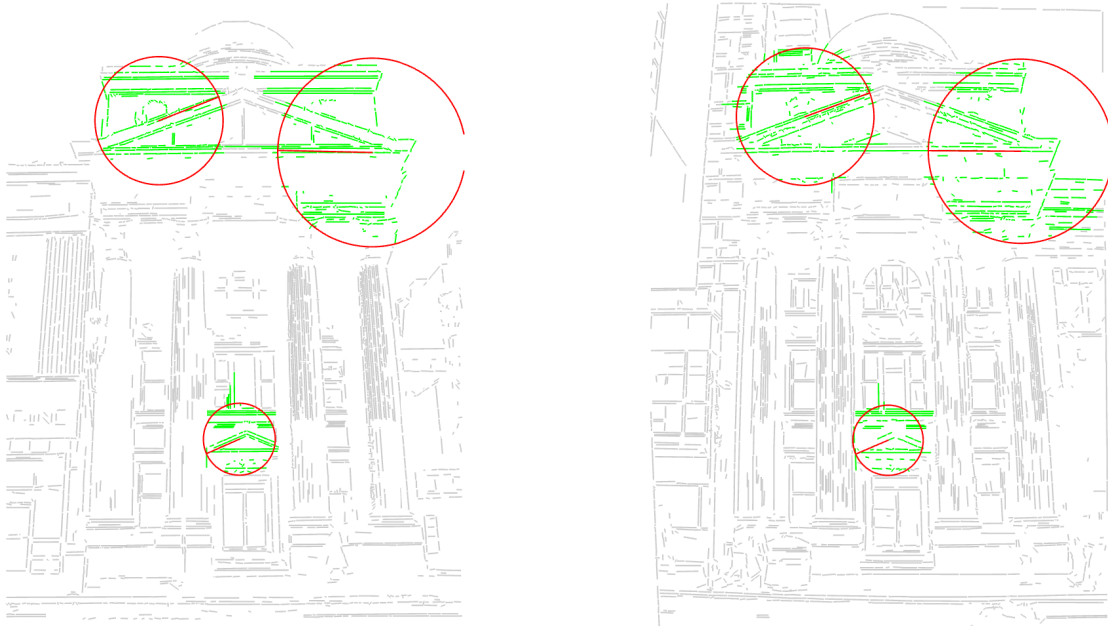


Figure 4.11. Three examples of correctly matched features and relevant line segments. A red circle represents a feature, and green line segments are those within a feature.

n_1 times. Third, noting that our descriptor operates in dual space, the $\mathbb{C}(S_i, S_j)$ check can be regarded as a confirmation process done in the primal space.

We illustrate three example matches in Figure 4.11. Each red circle represents a feature. Green line segments are the ones within a feature. Using dual parameters, we encode DUDE descriptors that capture the distribution of line segments relative to feature points. They are matched based on the similarity of their DUDE descriptors, if the matches are preserved throughout k trials. Optionally, one can remove those matches whose line similarity scores are less than the threshold ρ .

4.3 Experimental Results

To evaluate the efficacy of our detection and description, we follow the same evaluation strategy of [Hauagge and Snavely \[2012\]](#) and [Bansal and Daniilidis \[2013\]](#), as well as using the same dataset introduced by [Hauagge and Snavely \[2012\]](#), which exhibits dramatic appearance changes.

4.3.1 Evaluating detections

Repeatability is a common measure indicating an ability for a feature detector to provide common feature points between image pairs, calculated as the fraction of the number of

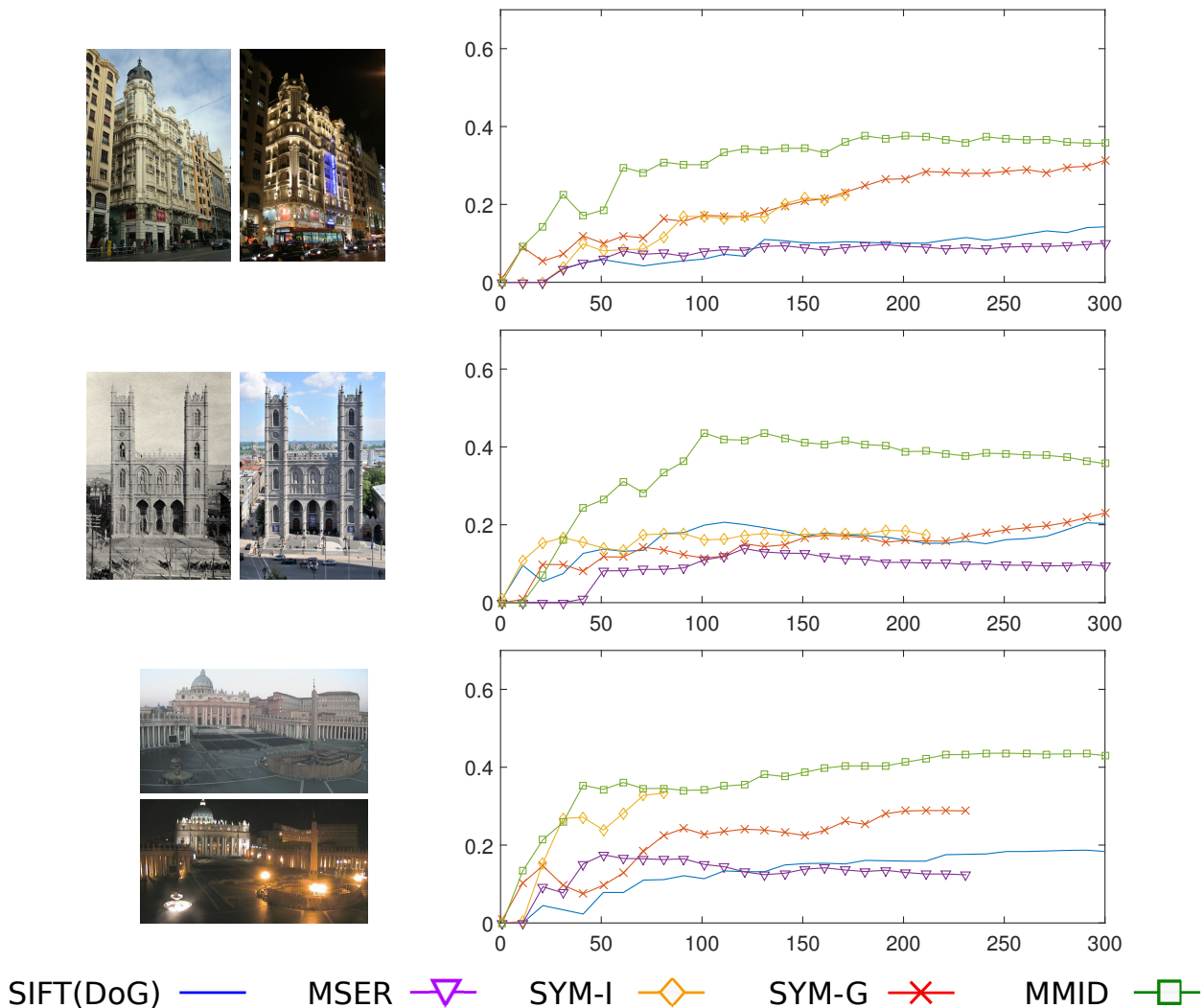


Figure 4.12. Repeatability curves. Five detection systems are compared: SIFT(DoG), MSER, SYM-I, SYM-G, and MMID (ours). For a given image pair (left column), repeatability (y -axis) is computed when considering top- k (x -axis).

repeatedly detected features over the total number of features.

To determine if $f_i \in F_1$ and $f_j \in F_2$ are repeatedly detected, the overlap measure introduced by Mikolajczyk et al. [2005] is widely used. In addition, because this overlap metric is more forgiving of larger features, the actual overlap computation is done after an additional normalizing step (see Hauage and Snavely [2012]). For the overlap measure, we set the same threshold, 0.6, as in Hauage and Snavely [2012] and Bansal and Danilidis [2013].

Note that simply producing a larger numbers of features can increase repeatability. Therefore the repeatability for evaluation should be computed and compared only by considering “top- k ” detections. To select a subset of k detections, one can sort features by scale fac-

	Scale		Score	
	100	200	100	200
MSER	0.087	0.103	-	-
SIFT (DoG)	0.144	0.153	0.050	0.078
SYM-I	0.135	0.184	0.173	0.206
SYM-G	0.173	0.228	0.227	0.281
JSPEC	0.287	0.292	-	-
MMID	0.217	0.310	-	-

Table 4.1. Repeatability

tors in decreasing order (larger features are preferred), or by some response scores (stronger features are preferred). However, like MSER or JSPEC, our detection does not define a response score, so we only use scale ordering.

We compare the performance of five detection systems: SIFT(DoG), MSER, SYM-I, SYM-G and MMID (ours). Figure 4.12 shows repeatability curves generated by varying k values (x -axis). We also provide the performance table suggested by Hauagge and Snavely [2012] (and also used by Bansal and Daniilidis [2013]) in Table 4.1, which indicates the average repeatability over the entire 46 image pairs when $k = 100$ and $k = 200$, respectively.

There are a few things to note. First, JSPEC requires significantly higher computation cost in time and space. Second, the repeatability should be understood carefully. Repeatability is only one aspect of feature detection, and it can be easily biased (e.g., to the number of detections or density of detections) and Table 4.1 represents only two cross-sections ($k = 100$ or $k = 200$). A detection with higher repeatability does not always mean a better input for feature descriptors. Because the final goal is to find a better set of correct correspondences, the evaluation should be also considered in combination with descriptors, as in the next section. Nevertheless, it is of course a good sign that our detection shows very good repeatability.

4.3.2 Evaluating descriptors

We conduct descriptor evaluation in a similar manner to Hauagge and Snavely [2012] and Bansal and Daniilidis [2013]. Each of the matches paired by descriptor similarity has a standard NNDR (Nearest Neighbor Distance Ratio) score [Lowe, 2004]. By varying the threshold on the NNDR score, and identifying which matches are correct (with a known ground truth transformation), we can obtain a precision-recall curve.

As in Hauagge and Snavely [2012] and Bansal and Daniilidis [2013], to evaluate the efficacy of detection and description separately, we compare the results from different combinations, as seen in the Table 4.2. The column headings list the different feature detection

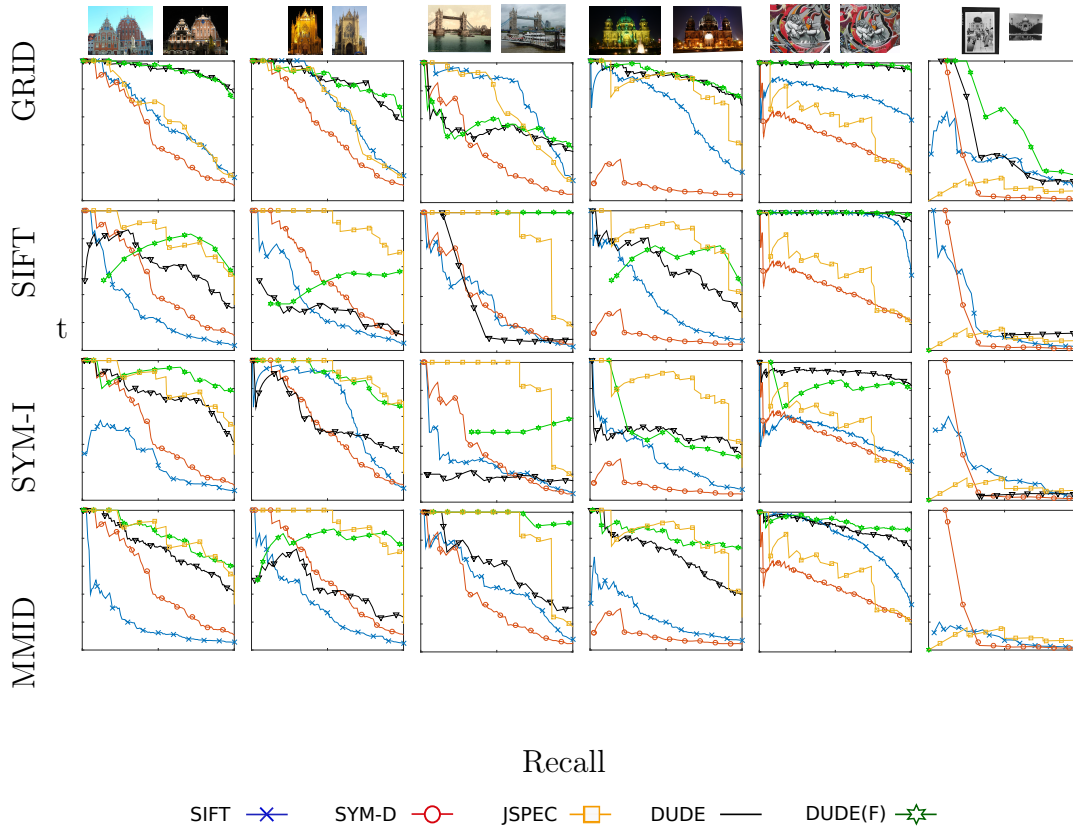


Figure 4.13. Precision(y)-Recall(x) curves comparing performance of descriptors: SIFT, SYM-D, JSPEC, DUDE, DUDE-F. The plots and example choices follow Bansal and Daniilidis [2013]. For each image pair (each column), and when using different feature detection types (each row), Precision-Recall curves are illustrated.

	GRID	SIFT	MMID	SYM-I	SYM-G	JSPEC
SIFT	0.49	0.21	0.24	0.28	0.25	0.61
SYMD	0.41	0.22	0.26	0.20	0.25	-
SIFT-SYMD	0.58	0.28	-	0.35	0.36	-
DUDE	0.63	0.35	0.42	0.40	-	-
DUDE-F	0.65	0.52	0.57	0.62	-	-

Table 4.2. Descriptor mean average precision (mAP) evaluation and comparison

types tested (GRID, SIFT [Lowe, 2004], MMID (ours), SYM-I and SYM-G [Haugge and Snavely, 2012], JSPEC [Bansal and Daniilidis, 2013]), and the row headings list the different descriptor types (SIFT [Lowe, 2004], SYMD and SIFT-SYMD [Haugge and Snavely, 2012], DUDE and DUDE-F (ours)). The detection type GRID is an artificially made-up feature [Haugge and Snavely, 2012]: for a set of feature points (e.g., every 25 pixels in x and y with angle 0° and scale 6.25) in I_1 , we remap the features onto I_2 with the actual known transformation. Therefore, GRID can be regarded as a perfect feature detector of repeatability 1, allowing evaluation of descriptor performance when detection is perfect.

In Figure 4.13, we illustrate precision-recall curves for a few image pairs. The choice of example image pairs is the same as in Bansal and Daniilidis [2013]. Each row represents a feature detection type. DUDE (black) and DUDE-F (green) show good performance overall except for the last image pair, where most of algorithms do not work well. However, when detection is perfect (GRID), DUDE and DUDE-F outperform their competitors. Because DUDE-F has one more step than DUDE, it is natural for DUDE-F to be better than DUDE.

Table 4.2 shows the summarized results by calculated mean average precision (mAP) over the entire set of 46 image pairs. Regardless of detection type, DUDE and DUDE-F outperform other descriptors. The combination of MMID \times DUDE-F show overall good performance, close to the state-of-the-art result of JSPEC. Because the source code of JSPEC is not publicly available, we could not test JSPEC \times {DUDE, DUDE-F}. We found SYM-I \times DUDE-F showed the best result, slightly better than JSPEC.

Note that we achieved comparable performance to JSPEC, only with much lower computation cost. DUDE processed these image pairs in minutes, and DUDE-F in tens of minutes using Matlab. To achieve its high performance, JSPEC requires eigen-value decomposition of a huge affinity matrix. For example, one needs to sample every 5 pixels and assign a 256D descriptor to each pixel. A $1K \times 1K$ image will generate 40,000 descriptors each of dimension 256. Then one needs to solve an eigen-value decomposition of an $80,000 \times 80,000$ matrix. Most of our calculations are simple matrix multiplications and/or simple iterations.

4.3.3 Shape matching using DUDE

In this section, we investigate DUDE for shape matching as an additional experiment. An example pair of input sets of line segments to be matched, S^P and S^Q , are illustrated in Figure 4.14. Note that the endpoints and angles of corresponding lines are intentionally not preserved and multiple additional line segments (noise) are added to S^Q . For each line segment $s_i^P \in S^P$ and $s_j^Q \in S^Q$, we formulate its DUDE descriptor (\mathbf{d}_i^P and \mathbf{d}_j^Q) with respect to its midpoint (reference location) and tangential angle (reference angle). In this subsection, we use $n_\theta = 12$ and $n_r = 5$ for the number of the DUDE histogram bins. We assign a matching cost (similarity distance) C_{ij} for a pair of descriptors, \mathbf{d}_i^P and \mathbf{d}_j^Q , using chi-squared (χ^2) histogram difference:

$$C_{ij} = \chi^2(\mathbf{d}_i^P, \mathbf{d}_j^Q) \quad (4.4)$$

where

$$\chi^2(\mathbf{s}, \mathbf{t}) = \frac{1}{2} \sum_k \frac{(s_k - t_k)^2}{(s_k + t_k)} \quad (4.5)$$

for two given histograms, each with k bins, $\mathbf{s} = \{s_k\}$ and $\mathbf{t} = \{t_k\}$. Note that one can represent the matching costs with a matrix $\mathbf{C} = \{C_{ij}\}$, which in general will not be square. In order to find the best match, one can solve the linear assignment problem using the Hungarian method [Kuhn, 1955; Munkres, 1957]¹, similar to [Belongie et al., 2002]. The algorithm solves for the one-to-one matching permutation π that minimizes the total cost

$$H(\pi) = \sum_i C_{i,\pi(i)}. \quad (4.6)$$

We use an efficient open-source Munkres method implementation [Cao, 2015], which takes a cost matrix \mathbf{C} as input. As in [Belongie et al., 2002], to deal with outlier line segments effectively, we add dummy nodes to each of the line segment sets S^P and S^Q respectively with a dummy cost ϵ . A line segment will be matched to a dummy node if all (available) matches with real nodes are more expensive than ϵ . The segments paired to these dummies are regarded as outliers. In our implementation, we set the number of dummy nodes as 25% of the number of actual nodes in each set, and use dummy cost $\epsilon = 0.25$. Unlike the algorithm used in [Belongie et al., 2002], the implementation of [Cao, 2015] does not require a square cost matrix.

The matching result is shown in Figure 4.14c. We visualize matching pairs using the same color and same number. The outliers are shown as gray lines with no number.

Next, to show rotation and scale invariance, we conduct a similar experiment on line segments detected from shape silhouettes, using an example with curved boundaries, that would not initially appear to be promising for a line-based approach. We illustrate with two silhouette pairs chosen from the Kimia99 dataset [Sharvit et al., 1998] that include slight appearance changes, as well as rotation (Figure 4.15a and Figure 4.15b), and scaling (Figure 4.15c and Figure 4.15d), respectively. To achieve rotation invariance, we set the frame angles of features (midpoints in these cases) as the orientation of the line segments. To achieve scale invariance, we set the histogram (outer) radius as a multiple (k) of the mean distance between all pairs of feature points, following [Belongie et al., 2002]. In the figure, we show the $k = 2$ radius circles we used for the given inputs.

In both cases, despite their differing endpoints, the DUDE descriptor can be used to find appropriate matches. The DUDE descriptor is robust against the existence of multiple short extraneous line segments such as arise due to noise, because extraneous line segments that are short will be culled.

¹Harold Kuhn named it the ‘‘Hungarian method’’ (1955) and James Munkres further developed the algorithm (1957). The algorithm is variously known as the Kuhn–Munkres algorithm, the Hungarian method, and the Munkres (assignment) algorithm.

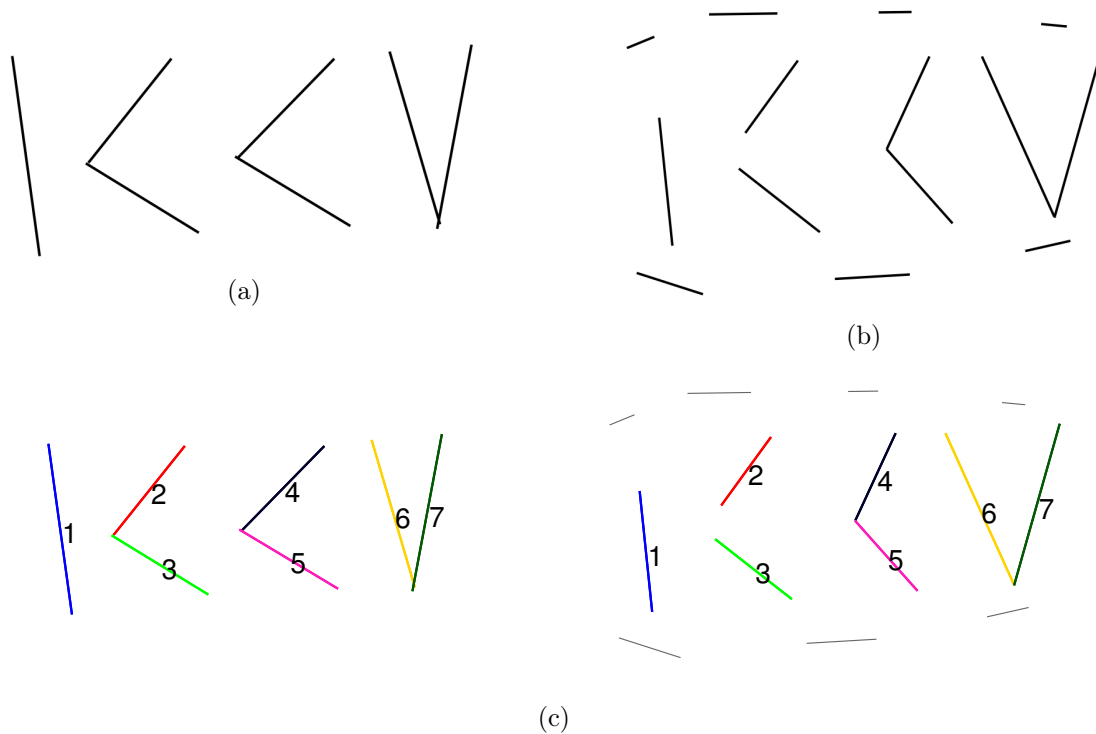


Figure 4.14. Matching of lines in the presence of unstable endpoint locations and noise: two sets of lines (a and b), and the matching result (c). Correspondences are shown both as the same number and color. Unmatched lines are shown in gray with no number.

4.3.4 Geometric characteristics of DUDE

We have not discussed a few geometric characteristics of DUDE, which might be possible for future research. Since DUDE is based on the $r - \theta$ dual space, DUDE description itself does incorporate rotational and translational information. Recall that usually rotational and translational invariance are acquired not from feature descriptors but from feature detections: once feature detectors examine feature points (as well as their scales and orientations) and hand over them to feature descriptors, feature detectors encode such given regions in an abstract manner.

Figure 4.16 visualizes the geometric patterns of DUDE representations of the input parallel and orthogonal line segments shown in red. In Figure 4.16a, parallel lines share the same θ values and different r values. Orthogonal lines have orthogonal θ values. Figure 4.16b shows how these patterns are preserved when feature orientations change. (The input line segments are the same as in Figure 4.16a.) When a feature orientation changes, it affects only θ s and preserves their relative angles to each other. Figure 4.16c shows patterns when feature positions change only via translation. When a feature translates, it affects only the r and f values. When determining matching descriptors, beyond just calculating descriptor

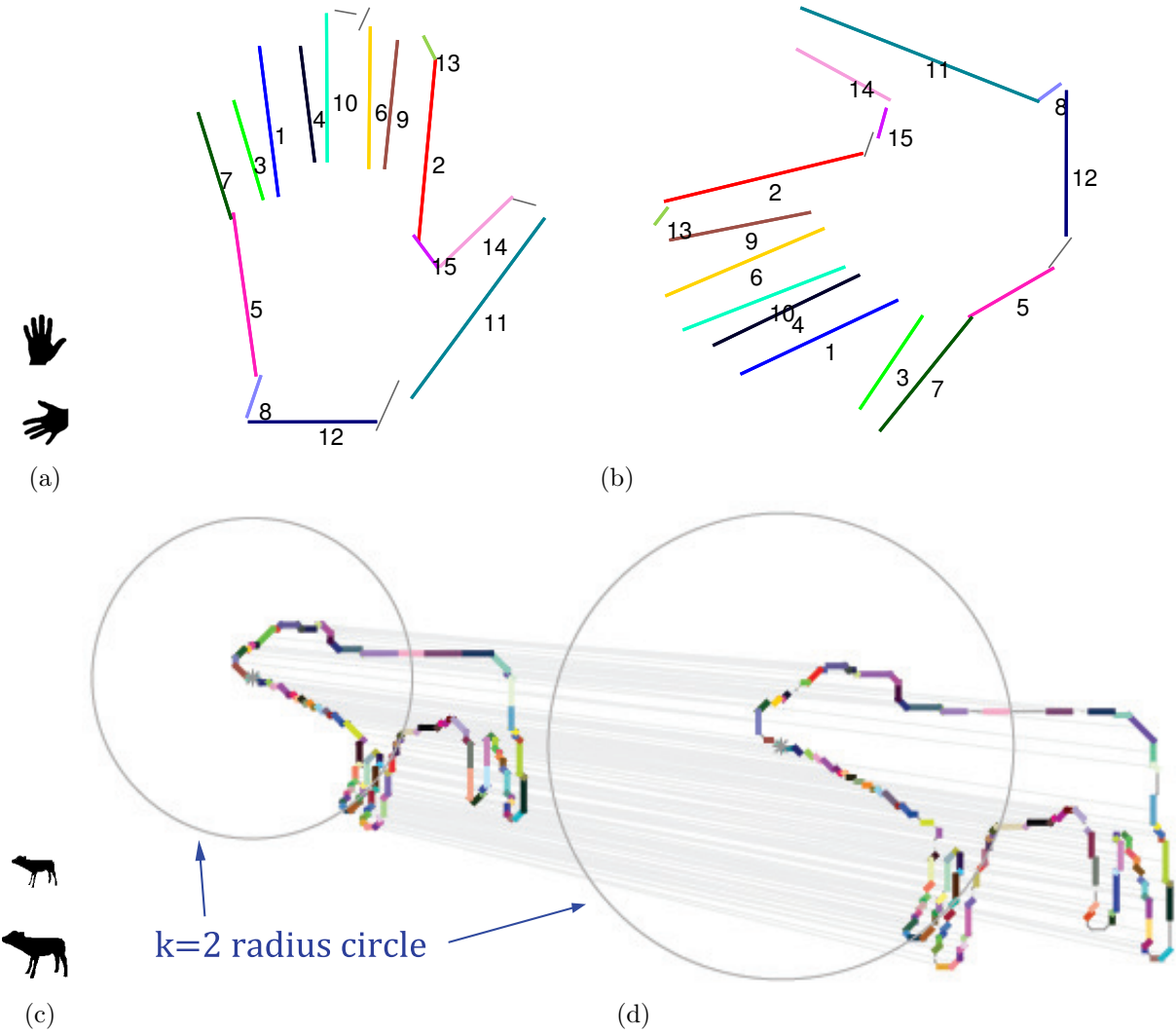
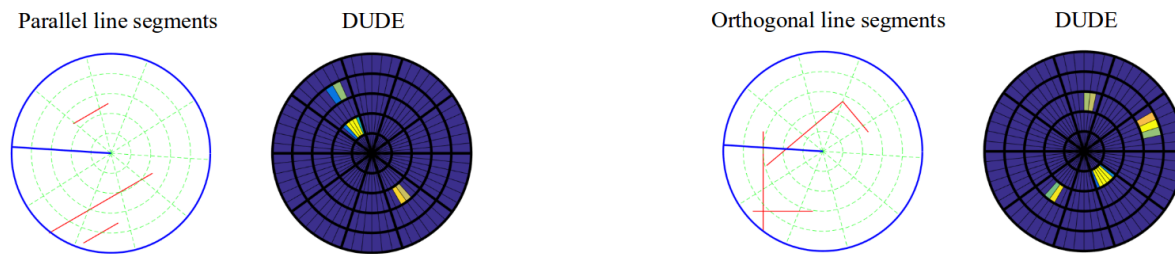


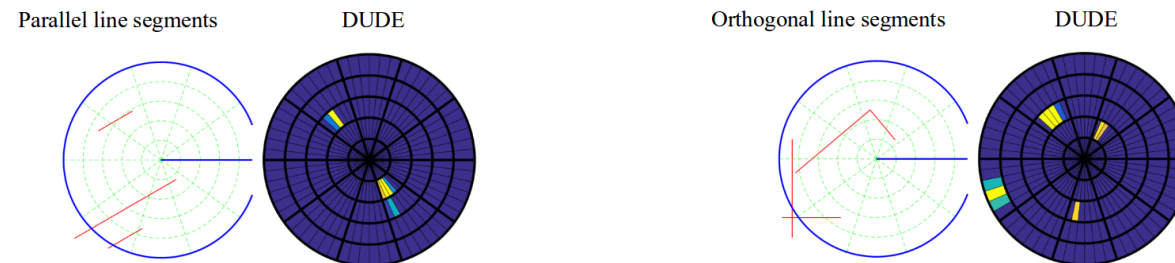
Figure 4.15. Rotation (b) and scale (d) invariance: [b] Colors and numbers mark matching pairs; line segment 8 is the only one that is not matched correctly. [d] Both colors and thin gray lines mark matching pairs. Circles show histogram radius. Note that there are different numbers of line segments detected.



(a) DUDE descriptors for parallel (left) and orthogonal (right) line segments shown in red. Parallel lines shares the same θ values and different r values. Orthogonal lines have orthogonal θ s. See Figure 4.6 for the details of this visualization scheme.



(b) DUDE descriptors for different feature orientation. Line segments are equivalent with (a). When a feature orientation changes, it affects only θ s while keeping relative angles to each other.



(c) DUDE descriptors for different feature location. Line segments are equivalent with (a). When a feature position changes, it affects only r and f values.

Figure 4.16. DUDE interpretations of the parallel and orthogonal line segments.

similarity from L_2 or other standard norms, one may be able to incorporate feature location or angles as well, which would not be possible with other existing descriptors.

4.4 Conclusion

We present a novel feature detector (MMID) and descriptor (DUDE) for challenging image pairs exhibiting dramatic appearance changes. In order not to be affected by such changes,

we extract information from line segments in images using a dual space of line segments. To acquire consistent line segments across disparate images, we conduct randomized hierarchical segmentation during feature detection and randomized perturbation during feature description. The experimental results show that DUDE descriptors significantly outperform most existing descriptors. In addition, DUDE shows equivalent performance to the state-of-the-art with significantly less computation cost.

Chapter 5

Autograder for Multiview Drawing Using DUDE

5.1 Introduction

In this chapter, we show an application of DUDE descriptors where shape comparison is required. Multiview drawing is an international standard “graphical language” to represent 3D objects with 2D drawings. By following the rules of the graphical language, people can communicate the shape of three-dimensional objects without ambiguity. A multiview drawing consists of orthogonal projections to mutually perpendicular planes, typically the *front*, *top*, and *right* views. In the U.S., these are arranged on the page using so-called third angle projection, as if orthogonal projections onto the sides of a transparent glass box containing the object had been unfolded onto the page [Bertoline et al., 2002]. The three typical projections of a simple 3D object under third angle projection are shown in Figure 5.2. Sometimes additional projections are drawn for interpretation convenience.

Due to the fundamental importance of engineering drawing for design and communication, engineering drawing is typically taught in a large class serving students majoring in

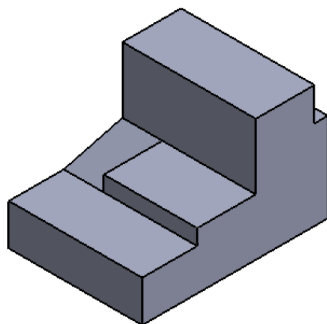


Figure 5.1. 3D geometry represented in multiview drawings in Figure 5.2

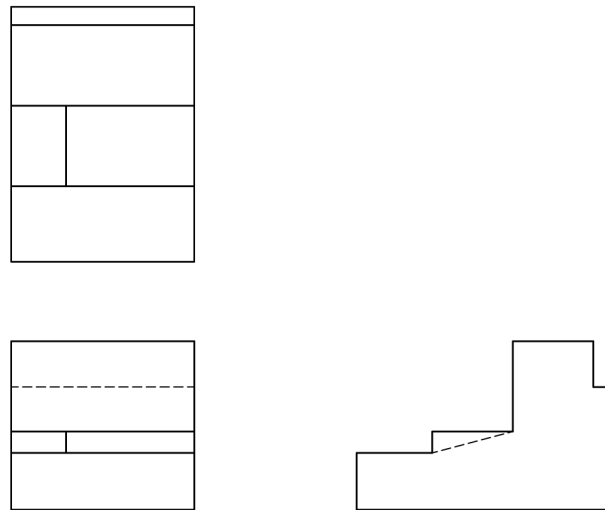


Figure 5.2. An example of a formal multiview drawing. Note that in multiview engineering drawings the views are not labeled; the placement and alignment communicates the relative viewpoints.

fields including mechanical engineering, electrical engineering, computer science, industrial engineering, civil engineering, nuclear engineering, and architecture in various universities. Some Massive Open Online Courses (MOOCs) that cover multiview drawing can be found in MIT Open Courseware [MIT Open Courseware, 2017] and Technical University of Madrid [Miriana, 2017].

Manually grading students' multiview drawing submissions and manually giving feedback to them is very time consuming, and the feedback is not always precise or timely. In the era of MOOCs, we expect high future demands for this type of engineering drawing course on an even larger scale, for which an automated grading/feedback tool would be critical. Particularly in a MOOC, but also with the large variety of backgrounds of students taking our on-campus course, different levels of students are engaged in the same curriculum. For effective education, we envision a system that should be able to distinguish them and provide specialized additional focused instruction and practice problems for different groups of students. To understand where students make mistakes frequently, an automated grading tool is essential not only for grading but also for analyzing "big data."

Our autograder addresses several frequent error types that inexperienced engineers and designers make [Lieu and Sorby, 2009] (Chapter 10), summarized below.

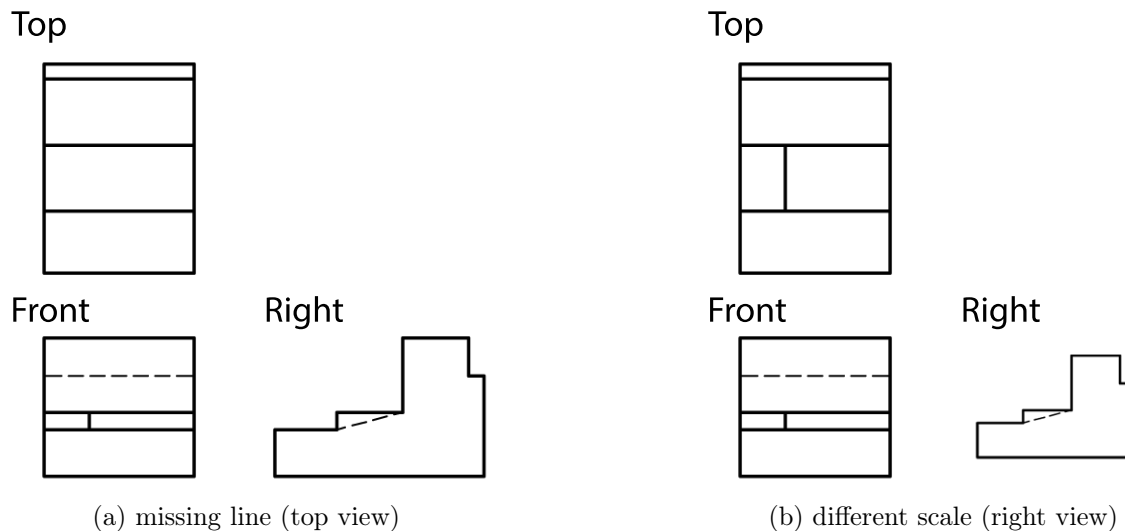


Figure 5.3. Two classes of mistakes. Note that the labels on the views are not present in the actual multiview drawing.

5.1.1 Missing and Incorrect Lines

A common problem with hand-created or 2D Computer-Aided Design (CAD) software-created drawings is that one or more lines may be missing. Figure 5.3(a) shows this error type. This error is especially difficult to recognize when someone else made the drawing; even when a grader has a solution to compare with, the grader may miss such a subtle mistake.

5.1.2 Mismatched View Scales

Each view of a drawing must have the same scale. Figure 5.3(b) shows an example when the scale of the right view is different, which makes for misaligned features between views. This is not permitted in multiview drawings. Note that as long as a drawing has the same scale throughout the views, the scale itself can be arbitrary for undimensioned drawings. So an automated grading tool should be *scale-invariant*, yet recognize mismatched scales between views in the same drawing.

5.1.3 Misaligned Views

Misaligned views, as shown in Figure 5.4(a), also make it difficult for a human to match up features between adjacent views; they are not permitted in multiview drawings. The orthogonal views must be aligned both horizontally and vertically. Note that once the views are aligned appropriately, the offset distances between pairs of adjacent views do not need to match. So an automated grading tool should be *offset-invariant*. Moreover, because the

entire drawing can be translated anywhere relative to the origin, the grading tool should be *translation-invariant*, up to alignment and relative location of views.

5.1.4 Views in Incorrect Relative Locations

Each view in a drawing must be located appropriately with respect to each other view. One possible mistake is caused by confusion of views (e.g., mistakenly placing a left view in the right view location). Sometimes students mistakenly rotate an entire view, typically by 90° . Another mistake is mirroring a view, as shown in Figure 5.4(b).

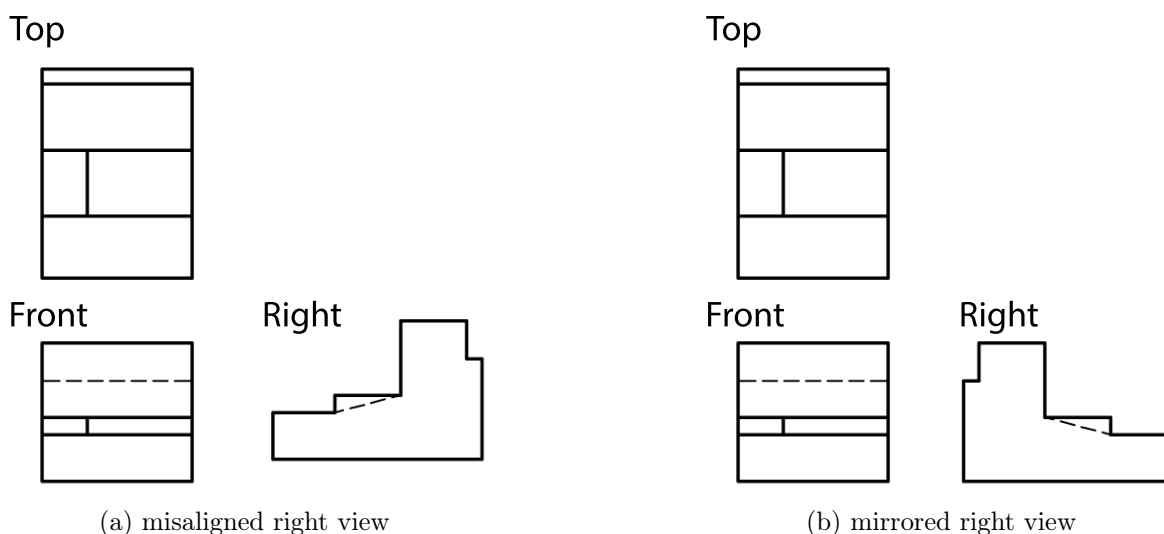


Figure 5.4. Two more classes of mistakes. Note that the labels on the views are not present in the actual multiview drawing.

These subtle mistakes are very easy for students to make, and are also easy for graders to miss. Especially with the traditional grading method where each student's printed drawing is graded by comparing it with a printed solution, a human grader can not guarantee a perfect comparison.

We show an example of a solution drawing and a student's drawing in Figure 5.5. Since they have different scale, translation, and offsets, the naïve comparison shown in Figure 5.5(c) does not work. Therefore we propose that an automated grading tool should be translation, scale, and offset-invariant when grading individual views, yet take these factors into account between views.

In this chapter, we propose a simple and flexible automated grading/feedback system, which is translation, scale, and offset-invariant in the sense described above. The proposed algorithm determines the transformation information for each view (top, front, and right)

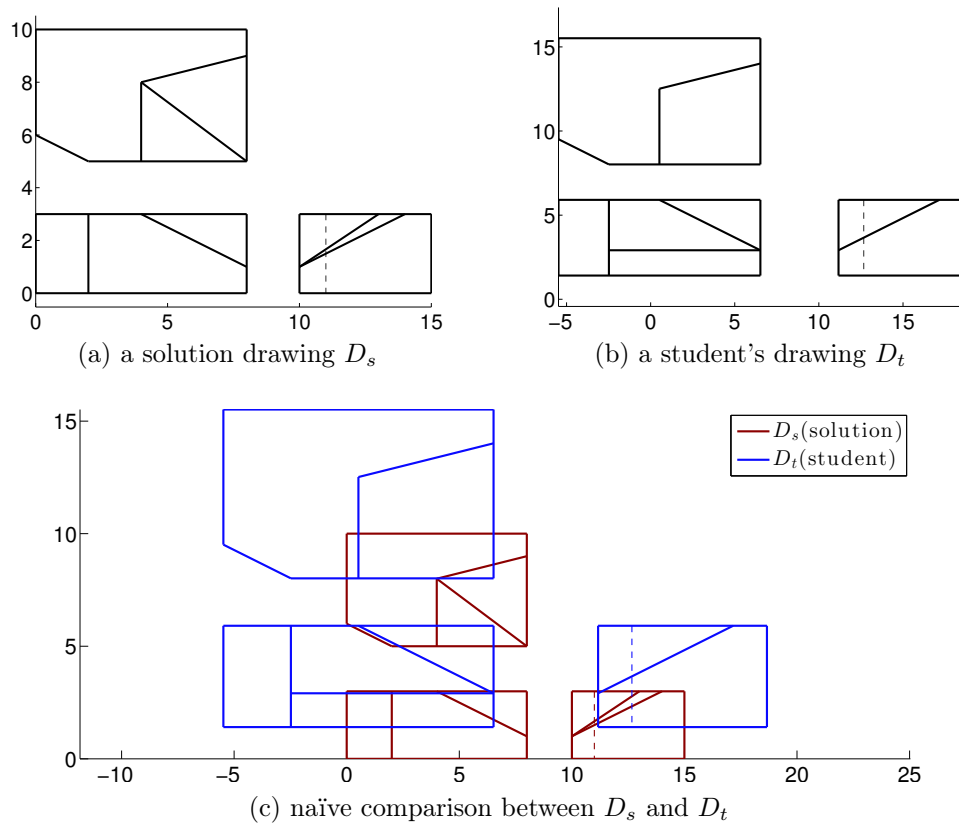


Figure 5.5. An example of (a) a solution drawing and (b) a student's drawings and (c) their naïve comparison. Because they have different scales, translations, and offsets, a naïve comparison does not work.

in a drawing (Section 5.3). We implement the automated grading/feedback system using MATLAB and address how the student errors detailed above can be graded using the transformation information (Section 5.4).

5.2 Related work

To our knowledge, our was the first work to address machine grading of multiview engineering drawings [Kwon and McMains, 2015]. AutoCAD provides a plug-in called Drawing Compare [AUTODESK, 2000], but it just visualizes the temporal changes of edits to a single drawing, and therefore it is not suitable to compare two drawings that include scale, translation, and offset differences.

There has been a fair amount of research on multiview engineering drawing interpretation in the context of using the drawings as input to reconstruct 3D models [Shin and Shin, 1998; Wang and Grinstein, 1993; Geng et al., 2002; Wang and Latif, 2003; Lee and Han, 2005].

However, none of these are useful to compare and grade multiview drawings, given that the reconstruction algorithms may fail when they face the incompleteness of students' drawings. Moreover the computation would be very intensive (as it would involve both reconstruction and 3D object comparison).

On the educational side, Suh and McCasland developed education software to help train students in the interpretation of multiview drawings [2009]. In their software, complete multiview drawings are given as input, and students are asked to draw the corresponding 3D models. This is very useful to enhance and evaluate students' multiview-drawings interpretation skills, the inverse of our purpose of evaluating students' multiview creation skills when 3D models are given as input. Since it is important for our students to be familiar with popular CAD software such as AutoCAD, for this work we chose to compare and grade native format AutoCAD files, which is easily extended to batch processing.

We use the random sample consensus (RANSAC) method [Fischler and Bolles, 1981] to estimate an affine transformation between the individual views of the two given drawings. RANSAC is an iterative method used to estimate parameters of a mathematical model from a set of data. RANSAC is very popular due to its effectiveness when the data has a high percentage of noise. The fact that much research in the computer vision field relies on RANSAC, for example, estimating the fundamental matrix [Brown and Lowe, 2005], recognizing primitive shapes from point-clouds [Schnabel et al., 2007], or estimating an affine transformation between two line sets [Coiras et al., 2000; Kwon, 2014], shows RANSAC's efficacy. There have also been many variations introduced such as MLESACK [Torr and Zisserman, 2000] and Preemptive RANSAC [Nistér, 2005], as well as research comparing the performance of the variations [Choi et al., 1997; Raguram et al., 2008]. In our current application, we have found that the original RANSAC concept is efficacious enough. We next discuss the basic RANSAC algorithm and how we apply it to estimate parameters of the transformation between single view drawings.

5.3 Algorithm

5.3.1 Single View Transformation Estimation

Initially we ignore the offset-invariant problem by assuming a drawing consists of only one view. Let V_s be a single view from the source drawing (solution), and V_t be a single view from the target drawing (student's). Then the task here is to estimate the optimal transformation T^* between V_s and V_t in order to address the translation and scale-invariant problems. Once we know this transformation, we can transform V_s into the coordinate system of V_t . Let V'_s be the transformed version of V_s . We denote this as

$$T^* : V_s \rightarrow V'_s$$

or equivalently,

$$V'_s = T^*(V_s).$$

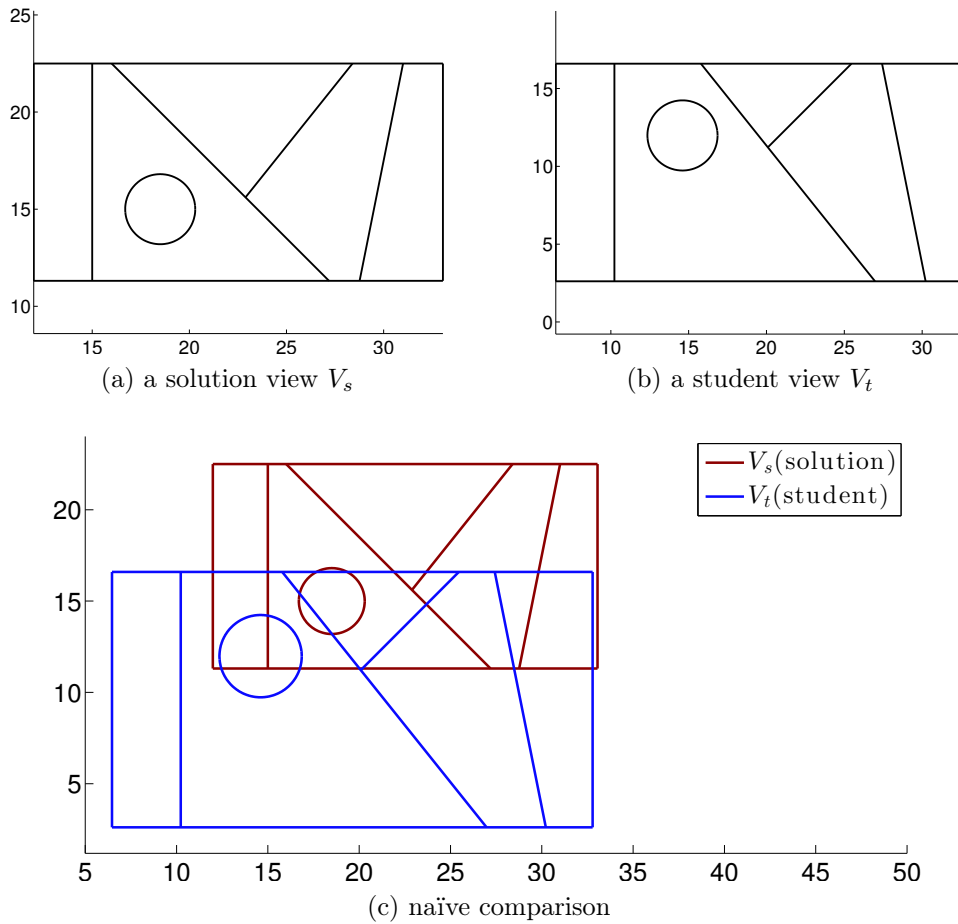


Figure 5.6. An example pair of views for transformation estimation.

We can then compare V'_s and V_t fairly. (In the next section, we will discuss how to apply this single view transformation in the context of full multiview drawings to address the flexible offsets permitted between views.)

As a transformation model between the two views, we assume an affine transformation. Its parameters are translation in x and y (t_x, t_y), scale in x and y (s_x, s_y), rotation θ , and skew k .

We take the pair of drawings in Figure 5.6 as an illustrative example for this section. V_t (Figure 5.6(b)) was obtained from V_s (Figure 5.6(a)) by applying a uniform scale, mirroring, and translation to V_s , and then editing two lines. It is not easy for a human to recognize what changes are there. The naïve comparison (Figure 5.6(c)) does not work at all. Even if scale and translation is properly considered, a grader may simply think most lines are slightly wrong as shown in Figure 5.7(a). However, better feedback can be provided by recognizing that the overall representation is in fact correct, except for mirroring and partially differing lines, as shown in Figure 5.7(b). Therefore, for a fair comparison that correctly identifies what

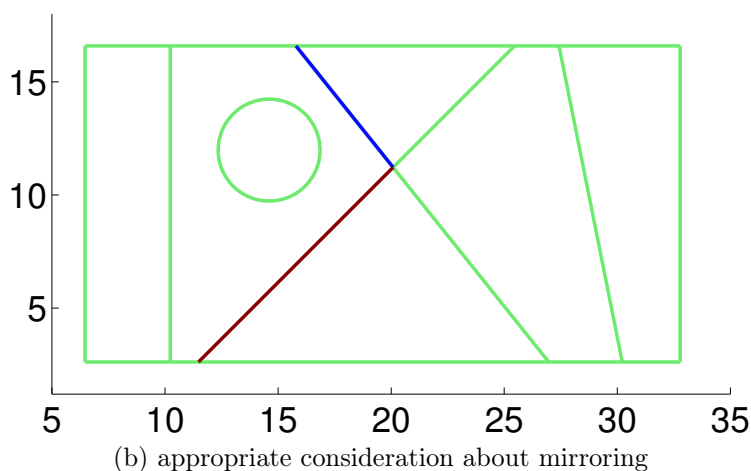
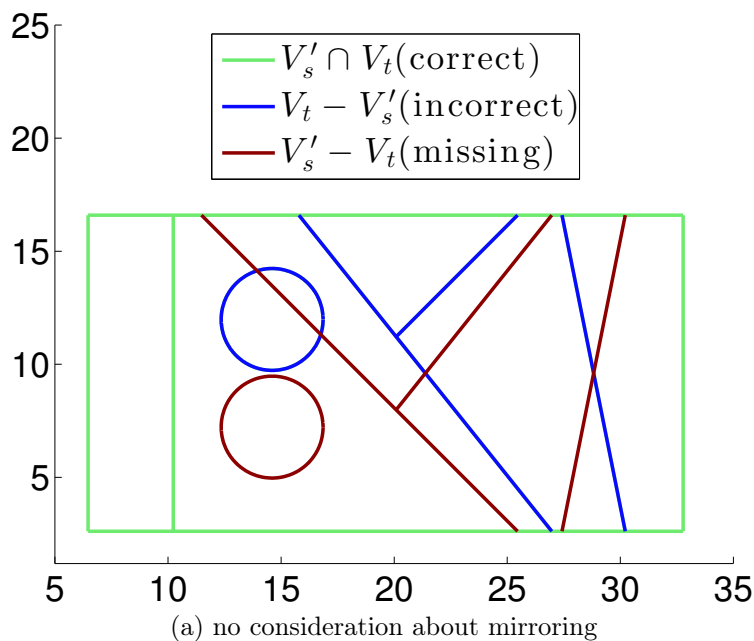


Figure 5.7. (a) Even if we align the two views in terms of scale and translation, it is not easy to compare them at a glance; here half the elements still appear to be slightly off. (b) In fact, most elements match perfectly if the correct affine transformation is applied. The real problem is mirroring and two lines that only partially differ.

conceptual errors led to the student’s mistake, we need to estimate the affine transformation and use it to align the two drawings first, before comparing individual line elements.

The affine transformation estimation procedure is based on RANSAC, which consists of the following generically described four steps:

1. At each iteration, randomly select a subset \mathcal{S} of the data set \mathcal{D} . Hypothesize that this subset (called *hypothetical inliers*) satisfies the ground truth model we seek.
2. Solve (or fit) the hypothetical model parameters Θ based on the hypothetical inliers \mathcal{S} . Note that \mathcal{S} is the only input for choosing Θ , so if \mathcal{S} includes incorrect “noisy” elements, naturally the estimated model parameters Θ will not be high quality.
3. Evaluate the estimated model parameters Θ using all data \mathcal{D} . The subset $\mathcal{C} \subseteq \mathcal{D}$ whose members are consistent with the estimated model parameters Θ is called a *consensus set*.
4. Iterate steps 1-3. The optimal choice of model parameters Θ^* is that with the largest consensus set. Terminate when the probability of finding a better consensus set is lower than a certain threshold.

Our data set \mathcal{D} is obtained by extracting certain points related to the elements in the drawings. The element types that we currently consider are *line*, *circle*, and *arc*. The point set consists of the two endpoints of line elements and the center points of circle and arc elements. Let P_s and P_t be the point sets extracted from all elements (lines, circles and arcs) from V_s and V_t , respectively. P_s and P_t together comprise the data set \mathcal{D} .

Since 2D affine transformations have six degrees of freedom (two each for translation and scale, and one each for rotation and skew), any three noncolinear point pairs (correspondences between the two views) will give a unique transformation. Therefore the hypothesis step of our RANSAC implementation is to choose three point pairs; our complete algorithm is summarized in Algorithm 1. The performance of the algorithm is highly dependent on the GETTHREEPAIRS function, i.e., the three point pair choices. One could just pick three random points from each of P_s and P_t , and pair them at every iteration, but that may require a huge number of iterations to come across a good choice. Alternatively, one can first conduct a priori point pairing between P_s and P_t based on their local appearance. In this case, whereas an advantage is that one only needs to choose three candidate pairs instead of 6 unpaired points to randomly pair at every trial, a risk is that there is a case that candidate pairs may not include three correct pairs of noncolinear points. We use DUDE for a priori pairing, and we show how much it can reduce computation time in Section 5.5.

In either case, the three randomly selected point pairs are the hypothetical inliers \mathcal{S} , and we solve for the (hypothetical) affine transformation matrix T based on the three pairs of points. The full 3×3 affine transformation matrix can be solved for using the homogeneous coordinate representation of the three pairs of points. (See for example [Hartley and Zisserman \[2003\]](#) for more details.)

Algorithm 1 Affine translation estimation for given two set of points using RANSAC

```

1: procedure GETAFFINE( $P_s, P_t, threshold$ )
2:    $s^* \leftarrow -\infty$ 
3:    $T^* \leftarrow \text{null}$ 
4:   while True do
5:      $p_s, p_t \leftarrow \text{GETTHREEPAIRS}(P_s, P_t)$ 
6:     if  $p_s$  is null or  $p_t$  is null then                                ▷ e.g., all three pairs are consumed
7:       break
8:     end if
9:      $T \leftarrow \text{SOLVEAFFINE}(p_s, p_t)$                                 ▷ solve with three chosen pairs
10:     $s \leftarrow \text{EVALUATEAFFINE}(T, P_s, P_t)$                             ▷ evaluate with all points
11:    if  $s > s^*$  then
12:       $s^* \leftarrow s, T^* \leftarrow T$ 
13:    end if
14:    if  $s^* > threshold$  then
15:      break
16:    end if
17:  end while
18:  return  $T^*$ 
19: end procedure

```

To evaluate T , we now transform the entire point set P_s by T . Let P'_s be the transformed version of P_s . If T is the optimal transformation, then most or even all points in P'_s will be coincident with those in P_t . We define the consensus set \mathcal{C} as $\mathcal{C} = P'_s \cap P_t$. Our evaluation metric is the cardinality of the consensus set (that is, the number of coincident points). We iterate this process; the optimal affine transformation T^* is the T with the largest $|\mathcal{C}|$. We can denote this as

$$\begin{aligned}
 P'_s &= T(P_s) \\
 T^* &= \arg \max_T |\mathcal{C}| \\
 &= \arg \max_T |P'_s \cap P_t| \\
 &= \arg \max_T |T(P_s) \cap P_t|
 \end{aligned}$$

where $\arg \max$ stands for the argument of the maximum, the argument element(s) of the given argument space for which the given function attains its maximum value.

We terminate the iteration when $|\mathcal{C}| > R * \min(|P_s|, |P_t|)$, where R is the minimum match rate, or all the cases are checked. We have found $R = 80\%$ to work well in practice. Once we have found a transformation that matches more than 80% of the points in the solution subview with points in the target drawing, we have found the region of interest that we are searching for, and there is no need to search further.

Consider the example of Figure 5.6; the optimal affine transformation is

$$T^* = \begin{bmatrix} 1.2494 & 0 & -8.5118 \\ 0 & -1.2494 & 30.7256 \\ 0 & 0 & 1 \end{bmatrix},$$

or equivalently, $t_x = -8.5118$, $t_y = 30.7256$, $s_x = 1.2494$, $s_y = -1.2494$, $\theta = 0$, and $k = 0$. Figure 5.7(b) shows the comparison between the transformed version of V_s , $V'_s = T^*(V_s)$, and V_t . In other words, we know that V_s should be scaled by 1.2494 and -1.2494 along the x and y axes respectively, and translated by $(-8.5118, +30.7256)$ in order to compare it to V_t . The opposite signs for the s_x and s_y scales indicates mirroring. There is no skew or rotation.

5.3.2 Application to Multiview Drawings

In this section, we discuss how to apply the transformation estimation process to multiview drawing grading. Again, let the source drawing D_s be the solution drawing and the target drawing D_t be a student's drawing.

First a grader must manually subdivide the solution drawing (but not the student's drawing) into the front, right, and top views. Call them V_{front} , V_{right} , and V_{top} , respectively:

$$D_s = V_{front} \cup V_{right} \cup V_{top}.$$

In the general case, a view can be any subset of the solution drawing. One can specify arbitrary views V_i depending on the complexity of the solution drawing:

$$D_s \supseteq \bigcup_i V_i.$$

We individually estimate optimal transformations $T_{V_i}^*$ between each view V_i ($\subseteq D_s$) and the entire student drawing D_t . By calculating separate transformations for each view, we can address the offset flexibility.

Consider the example input shown in Figure 5.5. For the front view, we have $t_x = -5.4785$, $t_y = 1.4114$, $s_x = 1.5$, $s_y = 1.5$, $\theta = 0$, $k = 0$. For the top view, we have $t_x = -5.4785$, $t_y = 8.0145$, $s_x = 1.5$, $s_y = 1.5$, $\theta = 0$, $k = 0$. For the right view, we have $t_x = 11.1657$, $t_y = 1.4114$, $s_x = 1.5$, $s_y = 1.5$, $\theta = 0$, $k = 0$.

We next discuss how these components, and their relationships, can be used to grade the student drawing.

5.4 Grading Checks

Once the optimal transformations $T_{V_i}^*$ (and their components) are calculated, one can set up a flexible set of rubrics. The checks described here correspond to the common student errors presented in the introduction.

5.4.1 Element Comparison

By applying each transformation $T_{V_i}^*$ to the corresponding view V_i from the solution D_s , we can compare individual elements of the two full multiview drawings. Suppose we have $T_{V_{front}}^*$, $T_{V_{top}}^*$ and $T_{V_{right}}^*$ from D_s . The transformed version D'_s is:

$$\begin{aligned} D'_s &= \bigcup_{V_i} T_{V_i}^*(V_i) \\ &= T_{V_{front}}^*(V_{front}) \cup T_{V_{top}}^*(V_{top}) \cup T_{V_{right}}^*(V_{right}). \end{aligned}$$

Figure 5.8 shows the transformed version of the solution, D'_s , super imposed on the student's drawing D_t . The transformed version has the same location, offset and scales as the student's. In Figure 5.8(c), red highlights the missed elements (elements that exist in the solution, but not in the student's drawing: $D'_s - D_t$), and blue highlights the incorrect elements (elements that exist in the student's drawing, but not in the solution, $D_t - D'_s$). If both set differences are the empty set, the two drawings are the same up to scale, translation, rotation, and skew.

5.4.2 Front-Right View Alignment

The front view and right view should be aligned horizontally. This can be checked by confirming that the t_y components of $T_{V_{front}}^*$ and $T_{V_{right}}^*$ are the same. We also need to check if the right view is still on the right side of the front view in the student's drawing. In other words, t_x of $T_{V_{right}}^*$ should be greater than t_x of $T_{V_{front}}^*$.

5.4.3 Front-Top View Alignment

The front view and top view should be aligned vertically. This can be checked by confirming that the t_x components of $T_{V_{front}}^*$ and $T_{V_{top}}^*$ are the same. We also need to check if the top view is still above the front view in the student's drawing. In other words, t_y of $T_{V_{top}}^*$ should be greater than t_y of $T_{V_{front}}^*$.

5.4.4 Uniform Scale

In multiview drawings, the aspect ratio must be preserved, and all views must have the same scale, even though the scale factor itself can be arbitrary. This can be checked by confirming that all six scale components (s_x and s_y of $T_{V_{front}}^*$, $T_{V_{top}}^*$, and $T_{V_{right}}^*$) are the same.

5.4.5 Mirroring

By confirming that the signs of all six scale components are positive, we can recognize mirroring, which should not be present.

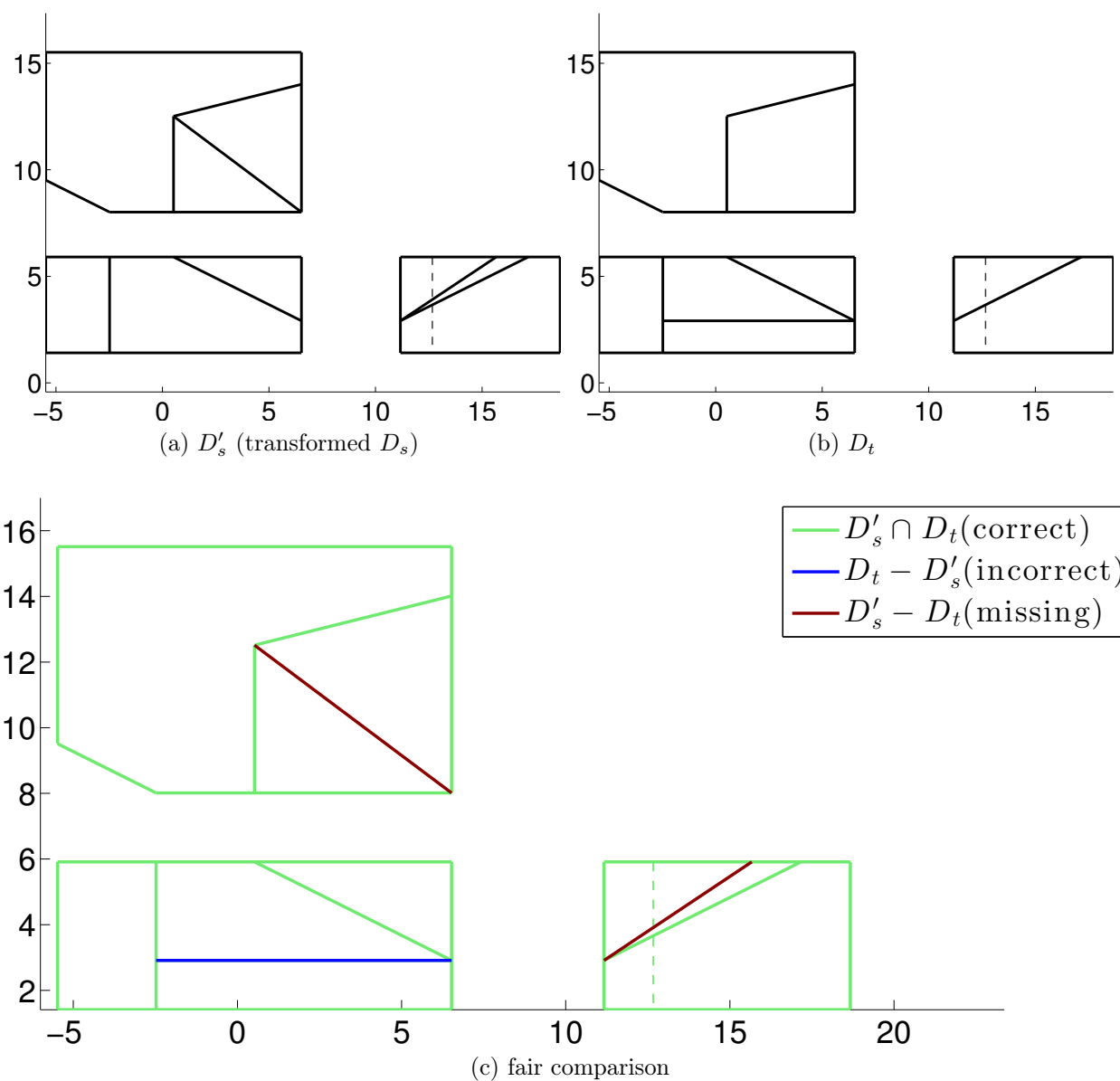


Figure 5.8. We estimate the transformation for each view individually using RANSAC. By applying the transformations to the views in D_s , we get the transformed version D'_s . Then the elements of D'_s and D_t can be compared one by one.

5.4.6 Rotation / Skew

The rotation and skew components of the transformations of all views should be zero, as long as the homework assignment is to reproduce the typical front, top, and right views.

5.5 Computational Improvement

Randomly finding three optimal point pairs may require a huge number of iterations. Let n_s and n_t be the cardinality of point sets P_s and P_t , respectively. In the hypothesis generation step in RANSAC, there are $6 \binom{n_s}{3} \binom{n_t}{3}$ possible cases, and for each case we need $n_s n_t$ comparisons to calculate the consensus set. First, we show how we can filter out some hypotheses to reduce computation in Section 5.5.1. Second, we conduct DUDE matching, and acquire candidate pairs, from which sets of three pairs are chosen. We introduce this approach in Section 5.5.2.

5.5.1 Filtering

5.5.1.1 Attribute and Vertex Degree Filtering

We store two simple attributes with each point: the element type ($\in \{line, circle, arc\}$) that gave rise to the point, and the number of intersecting elements at the point (in the case of the center points of circle and arcs, the number is zero). In the hypothesis generation step, we discard a hypothesis if these attributes are inconsistent for any of the pairs.

5.5.1.2 Transformation Filtering

Because the hypothesis transformations are acquired from the randomly chosen set of three pairs of points, most of them imply severe distortions, which are not typical of student errors. We can skip the evaluation step for this kind of unrealistic transformation. To filter out these cases, when we solve for T , we decompose T into the six components (DOFs). The unrealistic cases include when the absolute value of translations are too big, scales are too big, too small, or too imbalanced, etc. We skip those where:

- translation: $|t_x|, |t_y| > 300$ (The parameter 300 relates to the default canvas size of AUTOCAD software. Students may start their drawing from an arbitrary location. The parameter should be large enough to address these arbitrary translations. For our situation, students' arbitrary translations were not greater than 300.);
- scale: $|s_x|, |s_y| < 1/3$ or $|s_x|, |s_y| > 3$;
- skew: $|k| > 0.1$; and
- rotation: no constraint.

Here the thresholds for t_x, t_y, s_x and s_y can be determined based on the entire bounding box comparison (i.e., considering difference between solution drawing and student drawing). Students' mistakes do not tend to affect the skew, so theoretically k should be always zero, but due to numerical errors, the k value may be a very small number, e.g. 10^{-8} . Note that these thresholds are used solely to reduce the search space, and one can shrink/expand the permissible ranges if analysis of a larger dataset indicates smaller/larger variations to be present in other students' drawings.

5.5.1.3 Connectivity Filtering

To reduce the search space, we first analyze connectivity and divide an entire drawing into several candidate views by grouping connected elements, since these are almost always part of the same view. For explanation simplicity, assuming the elements of a drawing (of n points) are equally divided between the three views (each of $n/3$ points), then the computation efficiency improvement can be easily seen since $\binom{n}{3} = \frac{n!}{(n-3)!3!}$ is much larger than $3\binom{n/3}{3} = 3\frac{(n/3)!}{(n/3-3)!3!}$ (e.g., when $n = 60$, the difference is 34,220 versus 3,420). For a given source view (in the solution drawing), we randomly choose a connected view and choose three candidate corresponding points only within the connected view (in the student drawing) in the RANSAC algorithm.

However, we can do even better. It's a waste of effort to try to find corresponding points in a view that is not actually corresponding (and therefore no meaningful transformation exists). So knowing view-to-view relationships beforehand would further improve computation time. We show this can be done using DUDE descriptors in the following section.

5.5.2 DUDE-based choice

Since multiview drawings generally include many lines, DUDE is a good choice as a descriptor. Recall that a DUDE descriptor is defined at a location (x, y) with an orientation θ and a scale s . The numbers of histogram bins, n_r, n_θ, n_f , determine the dimension of descriptors. For each endpoint $p = (x, y)$ of each line l in a multiview drawing, we create a descriptor at the endpoint location (x, y) with a scale as the length of l . We always set descriptor orientation to be 0 since in multiview drawing, we assume rotation rarely happens. (If rotation is expected, one could take the orientation of l .) We visualize an example descriptor frame in Figure 5.9, where it is generated for the upper endpoint of a vertical line. Then, we match endpoints based on descriptors. The example matching result is shown in Figure 5.10. Note that in this example, we spatially spread out views in the target drawing (in blue) to show matching results. Once we conduct matching, we need to conclude which view in the solution corresponds to which view in the student drawing (even though the example in Figure 5.10 seems obvious, matching can be noisy for complex drawings.) We count the number of matches $n_{(i,j)}$ that correspond between view v_i^s view v_j^t and use the Munkres assignment algorithm [Munkres, 1957] (also known as the Hungarian algorithm) to choose the best match.

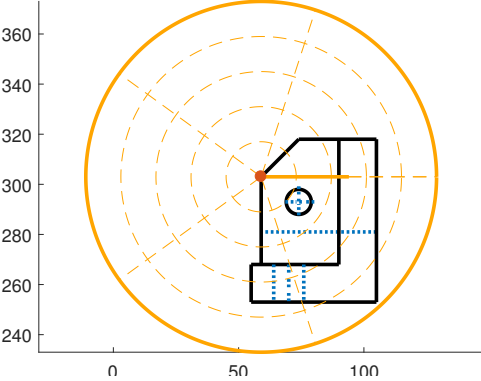


Figure 5.9. DUDE descriptor frame for a given point.

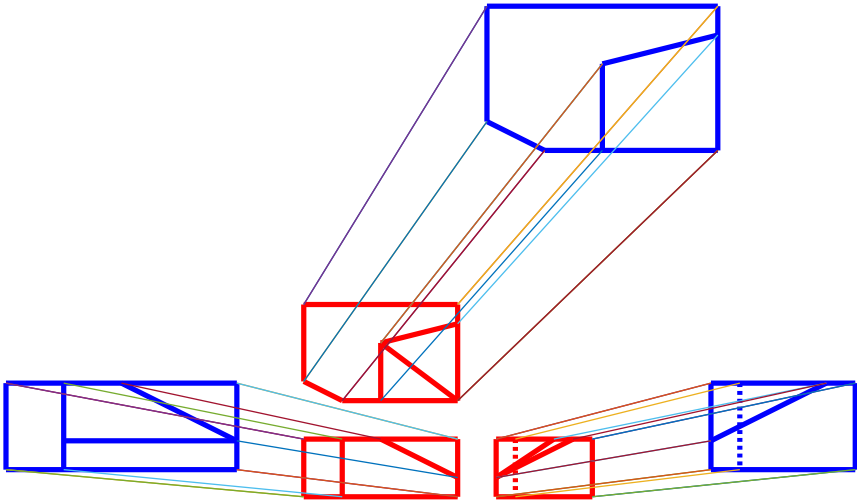


Figure 5.10. DUDE matching

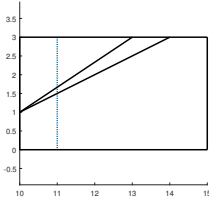
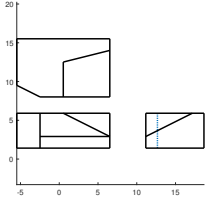
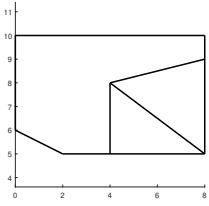
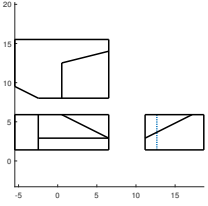
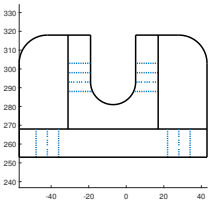
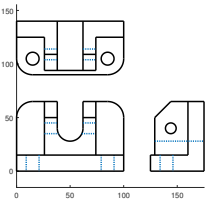
Input	Random-Filtered	DUDE
 	0.051 s	0.250 s
 	0.592 s	0.286 s
 	hours	0.448 s

Table 5.1. Computation time comparison.

With the view-to-view relationship, we re-conduct DUDE matching within corresponding views, and then run our RANSAC algorithm from Algorithm 1. The difference is that the earlier pair of random three-point choices in the hypothesis step is now random three-match choices. We show computation time comparison for three example views in Table 5.1. Since RANSAC is a randomized method, we ran it five times and averaged computation time. One can see that the computation time of the RANSAC algorithm without DUDE diverges quickly as the number of points increases (even with the filtering schemes) whereas the computation time of the DUDE-based method stays relatively stable. Note that since a pure MATLAB implementation of the the RANSAC portion was too much slow, we convert the portion into a MEX (C/C++ code subroutine call) implementation, whereas we kept the MATLAB implementation of the portion for the DUDE-based method. Therefore, the actual time difference is even greater than shown in Table 5.1.

5.6 Implementation Issues

In practice, there are additional steps that needed to be implemented to fully automate the grading/feedback system. We briefly mention some of them below.

5.6.1 Converting from DWG to DXF

Students in the class draw using AutoCAD, which by default saves files in DWG format. Because AutoCAD is commercial software and DWG is its binary file format, to our knowledge, there is no open source code for accessing DWG files directly. So we need to convert DWG files to DXF file format, which is a file format developed by AutoDesk for enabling data interoperability between AutoCAD and other programs. For an automated batch process of this conversion on all students' submission, we also implemented an AUTOLISP script, which runs in AutoCAD.

5.6.2 Loading DXF in MATLAB

We extract drawing elements from each DXF file using MATLAB. Currently the loading operation is based on the open source code [Mathworks, 2000b] in the MATLAB CENTRAL File Exchange website [Mathworks, 2000a].

5.6.3 Merging Elements

Some elements may be drawn (partially) duplicated, overlapping, or decomposed into sub-segments. Especially in the case of lines/arcs, they may consist of several connected or overlapping partial lines/arcs instead of one long line/arc. For this reason, we merge objects into one if they can be represented as a simpler one. This also makes the point set smaller, which reduces computation time.

5.6.4 Pre-defining Layer Names

Currently we do not autograde dimensioning and header parts of multiview drawings, only visible and hidden lines. Since visible and hidden lines should be drawn with different line styles and thickness, we teach students to put them in separate layers and define these properties to apply to the entire layer. For autograding, we provide a template with the layer names, and only load elements drawn on the “visible” and “hidden” layers. Even though giving predetermined layer names is a constraint for the autograding system, declaring layers and grouping objects of the same type into a single layer is an important skill for student to learn regardless.

5.7 Results

We show another grading example in Figure 5.11. The solution drawing D_s (Figure 5.11a) and the student drawing D_t (Figure 5.11b) can not be compared using a naïve algorithm due to translation, scale, and offsets (Figure 5.11c). Using the estimated transformation for each view, we take our transformed version of the solution, D'_s , and compare D_s and D_t (Figure 5.11d).

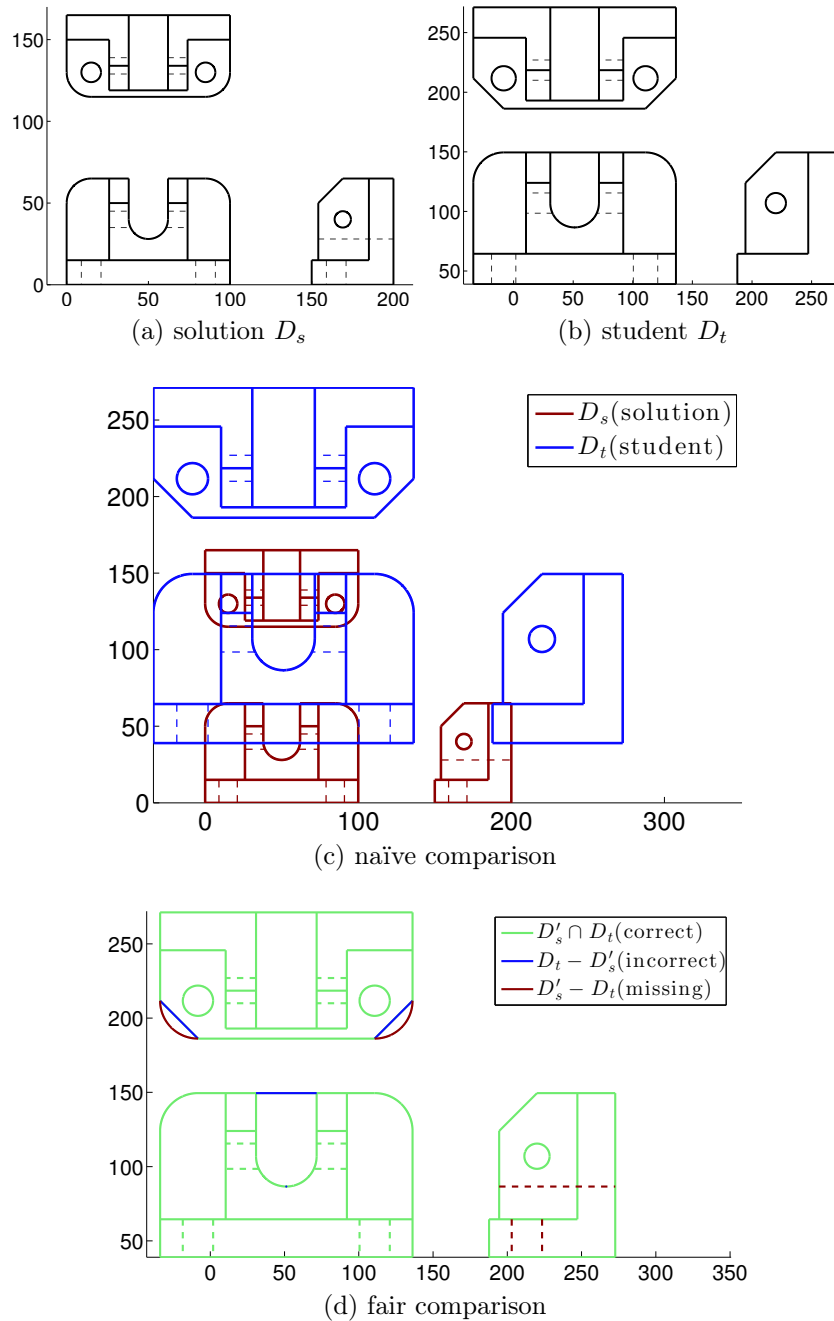


Figure 5.11. Comparing the solution to a student's drawing. To be compared to D_t , all views in D_s are scaled 1.7 times larger. The top, front, and right views are translated $(-33.8, +186.2)$, $(-33.8, +39)$, and $(+187.7, +39)$, respectively. All views have zero rotation and skew. By aligning them, the algorithm finds incorrect and missing lines, which are represented in blue and dark red.

5.7.1 Grading result visualization

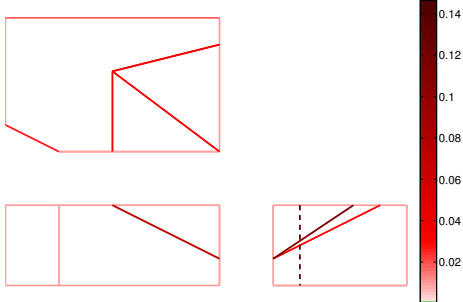
Beyond the advantages of more accurate, timely feedback to students, another advantage of an autograding tool will be its ability to analyze and summarize the grading results. As an example, we can visualize which elements of a drawing were most frequently drawn incorrectly by students, which can be useful information for instructors. We ran our algorithm in batch mode on the submissions in Fall 2013 for two problems assigned in homeworks #2 and #3 in E28. These assignment batches consisted of 115 and 113 students' submissions respectively. For each element in the solution drawing, we count in how many student submissions it is "missing." Similarly, for each "incorrect" element in the student drawing, we count how many student submissions have it. Figure 5.12 shows the solution with the elements color coded: the most difficult elements — that are most frequently missing/incorrect — are represented in dark red/blue, and those less frequently missing/incorrect are represented in light red/blue. In the problem from assignment #2, we can see that the top view causes more mistakes than the other views, and that students miss the diagonal and hidden lines in the front and right view most frequently (Figure 5.12a). Figure 5.12b shows that the diagonal features are frequently misdrawn. In the problem from assignment #3, many times students get confused in the upper part of the front view, and hidden lines are frequently missed.

5.7.2 Comparison with human grading

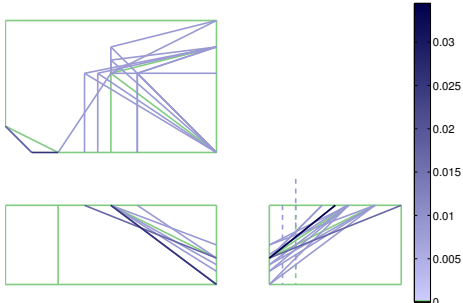
To verify the efficacy of the proposed algorithm, we compare the autograding results with human grading. A human grader with a full semester of experience grading for the course graded the 115 submissions of the homework #2 problem introduced above, using grade-scope [Gradescope, 2017] with pdf files of the submission.

We divide the comparison results into four categories (Figure 5.13). In the case of category A and B, autograding and human grading find the same errors, which account for 74.8% of the total 115 submissions. In the case of category B, although the same drawing elements are identified as errors, the human grader described them differently in her grading feedback to the student. Figure 5.14 shows two examples of category B. While the human grader interprets the mistake as "lines not aligned," the autograder reports it as the number of missing lines and incorrect lines. The human's interpretation can be more flexible, nuanced, and higher level. We leave more advanced emulation of such human grading rubrics as future work.

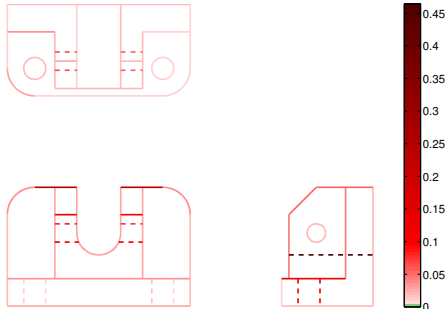
The most dramatic result is category C. For 21.7% of the submissions, the new autograding system catches students' mistakes that the human grader misses. We show two examples in Figure 5.15. This happens especially when a drawing includes subtle mistakes such as a slightly incorrect location, and/or when a drawing includes incorrect locations that are nonetheless consistent in neighboring views. In these cases, human graders may not notice them on the printed drawing.



(a) visualization of missing errors on a problem from assignment #2



(b) visualization of incorrect errors on a problem from assignment #2. Correct lines (solution) are shown in green.



(c) visualization of missing errors on a problem from assignment #3

Figure 5.12. Color coded difficulty. The elements that are most frequently “missing” are shown in dark red, and those less frequently missing are shown in light red (a and c). The elements that are most frequently “incorrect” are shown in dark blue, and those less frequently incorrect shown in light blue (b). The numbers in the color bar indicate the fraction of student submissions that made the mistake for each element.

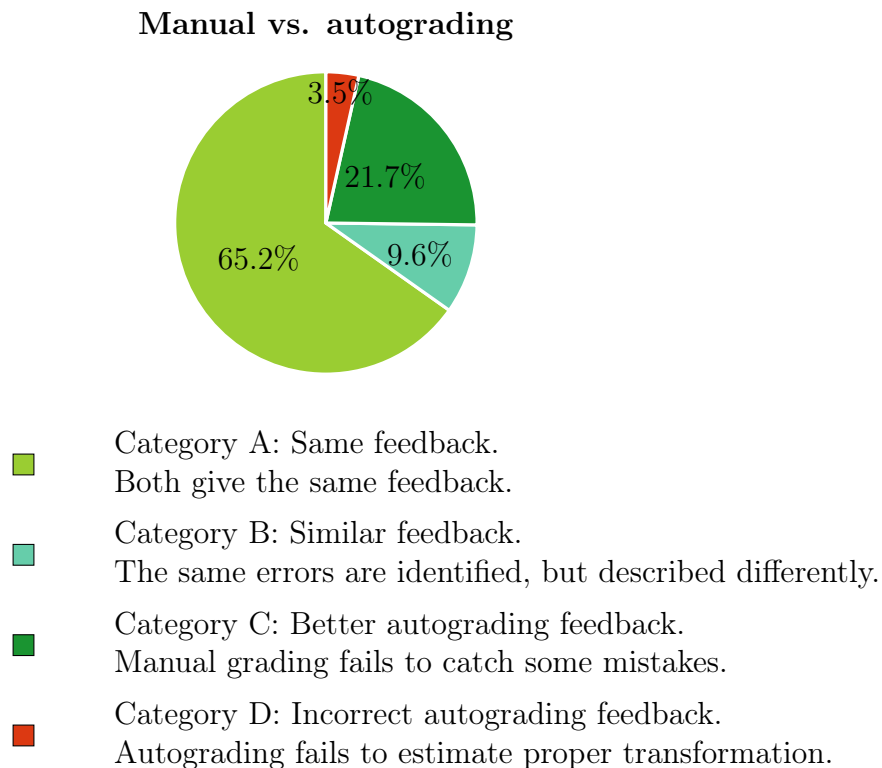
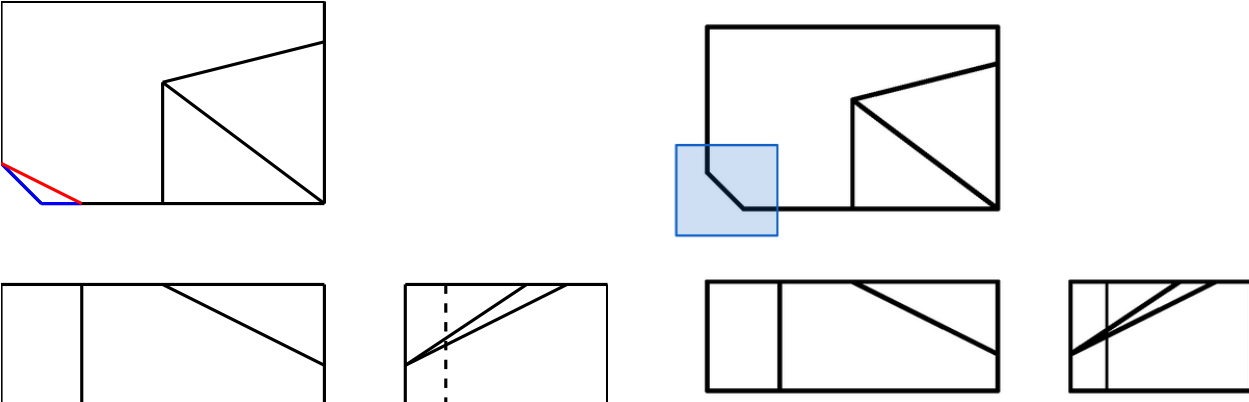


Figure 5.13. Comparison with human grading results.

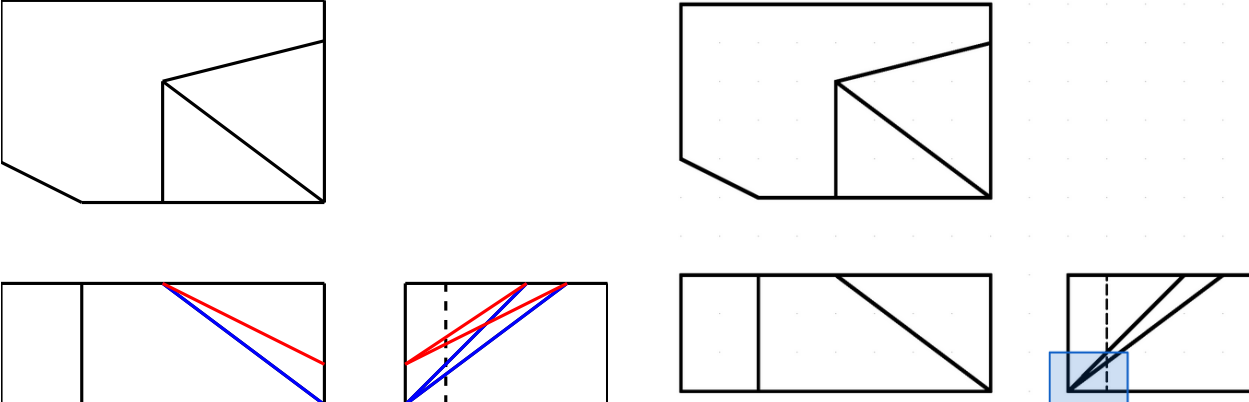
For 3.5% of the submissions, the autograding system failed to estimate the appropriate transformation, and gave incorrect feedback. We show an example in Figure 5.16. The student drawing (Figure 5.16b) has an incorrect front view. Note that our RANSAC evaluation metric is the number of coincident points. When a student drawing has several wrong elements, the RANSAC algorithm may regard a strange transformation as the best transformation for the reason that it yields the maximum number of coincident points. We expect that this problem can be solved by extending the RANSAC evaluation metric to consider lines as well as points in future work. Note that even though the proposed algorithm fails to give correct feedback for 3.5% of the submissions, this happens only when the student drawing has multiple errors. In these cases, a whole view was reported as incorrect by the autograder, where in fact some partial credit should have been given. The proposed algorithm never failed to recognize perfect drawings as such.

5.8 Conclusion

Motivated by the importance of an automated grading system for multiview drawings, we propose a novel system that can compare two multiview drawings, a reference solution and a student's submission, that may have inconsistent translations, scales, mirroring, skew, and/or



(a) While autograding (left) reports “1 missing line, 2 incorrect lines”, a human grader (right) reports “1 incorrect position.”



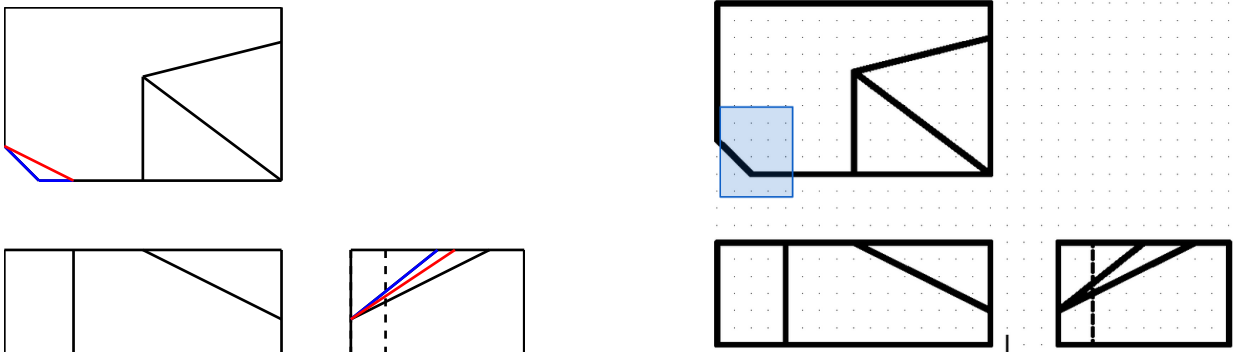
(b) While autograding (left) reports “3 missing line, 3 incorrect lines”, a human grader (right) reports “1 line not aligned.”

Figure 5.14. Two examples of category B. Even though different rubrics are applied, errors are identified.

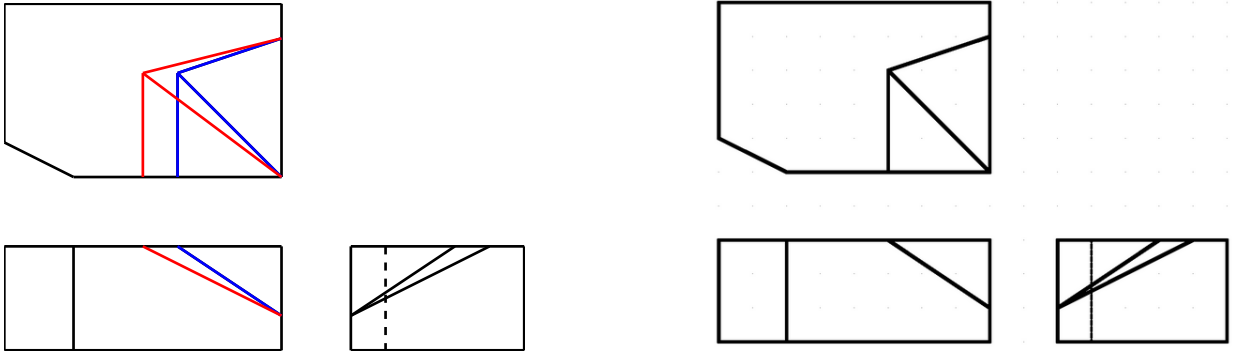
rotation; all of which must be distinguished from allowable differences in scale, offset, and translation to reliably identify and classify errors in the students’ drawings.

Our system provides fair comparison and grading checks for students’ drawings, which can be used as input for a flexible scoring system. For example, in many cases, a grader may not want to reduce scores for duplicate errors of the same type. A grader may want to place different scores (emphasis) on different elements. One possibility would be to use our element-wise difficulty analysis to assign an appropriate score to each element.

The proposed system can be useful for large classes, eliminating time consuming manual grading and incomplete feedback, and for MOOCs on engineering drawing, which currently do not exist.



(a) A human grader (right) succeeded in recognizing the incorrect elements in the top view, but failed to catch the element in a slightly incorrect position in the right view.



(b) A human grader (right) failed to notice elements in incorrect positions and gave a perfect score. Such errors are difficult for humans to catch because the positions are consistent with the neighboring view.

Figure 5.15. Two examples of category C. While a human grader failed to notice these mistakes, our autograding system found them.

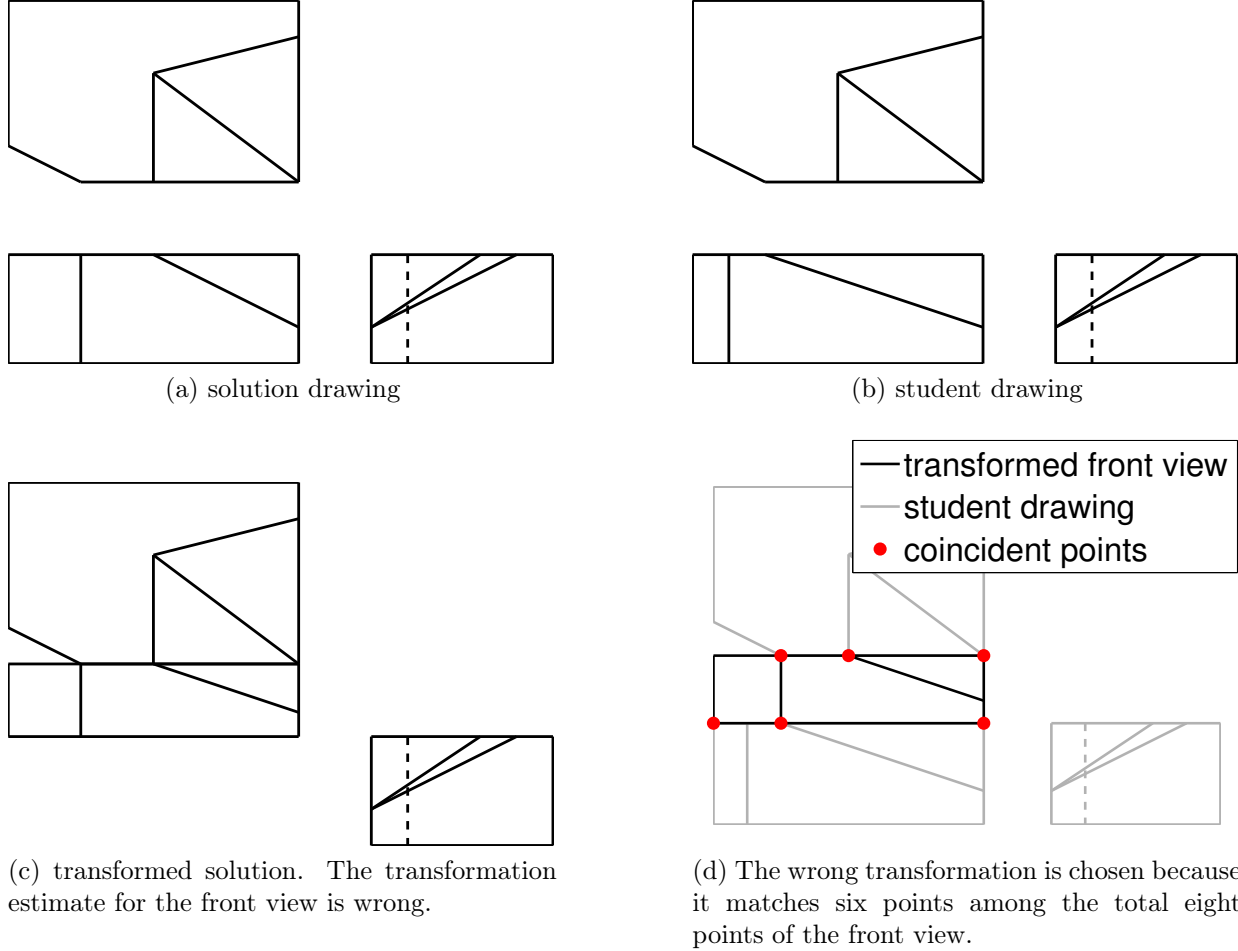


Figure 5.16. An example of category D. Our algorithm failed to estimate an appropriate transformation for the front view of the solution drawing. Note that the student drawing (b) has multiple mistakes in the front view.

Chapter 6

Deep Learning Based Image Matching

6.1 Introduction

In previous chapters we discussed the DUDE descriptor, designed to capture the distribution of line segments, and its use for image registration. Alternate approaches to descriptor design that have recently become popular are based on deep learning (data-driven) methodologies [Simo-Serra et al., 2015; Zagoruyko and Komodakis, 2015; Kumar et al., 2016; Lin et al., 2016; Balntas et al., 2016; Yi et al., 2016]. However, an obstacle to applying such methodologies to “learn” descriptors that will be effective for registering pairs of disparate images is the lack of sufficiently large disparate image datasets providing the amount of labeled data that deep learning approaches require. We here discuss a new approach to exploiting deep learning to develop descriptors that will be effective on disparate imagery, even in the absence of training data that exhibit the dramatically different appearance of the disparate imagery we ultimately hope to register.

For better generalization, we introduce a new data augmentation strategy, *Artificial Intensity Remapping* (AIR). We aggressively generate additional challenging but realistic input for training as illustrated in Figure 6.1. To demonstrate the efficacy of AIR, we train a model in a manner similar to Simo-Serra et al. [2015] with AIR, and evaluate it with disparate test sets, which would normally be considered to be adversarial input. For this purpose, we collect and build disparate test sets from various sources [Kelman et al., 2007; Hauagge and Snavely, 2012; Razakarivony and Jurie, 2015; Rochester Institute of Technology, 2016; Visual Geometry Group, University of Oxford, 2016; Sandia National Laboratories, 2016]. Our experiments show that AIR is effective not only for standard test cases set but also for disparate test cases.

We also compare (hand-engineered) DUDE and (data-driven) AIR experimentally. Our findings are in line with that of a recent comprehensive evaluation done by Schönberger et al. [2017] in that advanced hand-engineered feature descriptors are similar or better than data-driven feature descriptors.



Figure 6.1. Examples of AIR: one original patch (leftmost) and its seven derived patches

6.2 Datasets

For our non-disparate input, we use the Multi-view Stereo Correspondence dataset (MVS) [Brown et al., 2011], which consists of 64×64 gray-scale image patches sampled from 3D reconstructions of the Statue of Liberty (LY), Notre Dame (ND) and Yosemite (YO), with a list of correspondences. Some corresponding patches are shown in Figure 6.2a–Figure 6.2c. For training, we select 300,000 positive samples (corresponding patch pairs with label 1) and negative samples (non-corresponding random patch pairs with label 0) for each subset of the MVS dataset (LY, ND, YO), following Simo-Serra et al. [2015]. We train a model with the sampled patches of two subsets (e.g., ND and YO), and test using the other subset (e.g., LY). The testing process is discussed in Section 6.4.

Despite training and test set separation, the three subsets may share a certain style of appearance. Overfitting to this style may occur during learning, and if so, the resulting descriptors may not work as well for input from different datasets.

We create two adversarial test sets (see Figure 6.2d–Figure 6.2e) by extracting patches from disparate image pairs. For the first set (MU1), we gathered 100 image pairs from various sources:

- 46 image pairs of various modalities, the complete set from Hauage and Snavely [2012];
- 5 image pairs of various modalities randomly selected from Kelman et al. [2007];
- 5 image pairs of years-apart aerial images, and SAR (synthetic aperture radar) aerial images randomly selected from Sandia National Laboratory [Sandia National Laboratories, 2016] vs. their correspondences obtained from Google Earth;
- 19 image pairs of temporally different aerial images randomly selected from the Rochester dataset [Rochester Institute of Technology, 2016];
- 25 image pairs, that although uni-modal have significant view point / illumination / etc. changes, selected from the Oxford dataset [Visual Geometry Group, University of Oxford, 2016].

For the second set (MU2), we use all 980 image pairs from EO (electro-optic) vs. IR (infrared) sensors from Razakarivony and Jurie [2015].

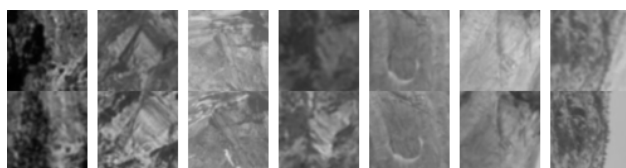
In contrast to the image patches of the MVS dataset, these disparate datasets consist of corresponding full image pairs with known ground truth transformations between pairs. For a given image pair, we registered the images using this transformation, and cropped 64×64 image patches located at every stride of 32 pixels. From the first image pair set (100 image



(a) Statue of Liberty (LY) from [Brown et al. \[2011\]](#)



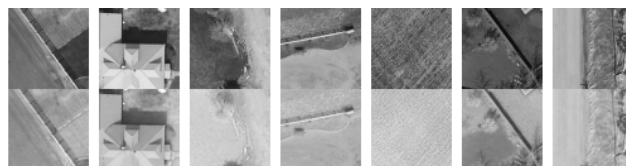
(b) Notre dame (ND) from [Brown et al. \[2011\]](#)



(c) Yosemite (YO) from [Brown et al. \[2011\]](#)



(d) Disparateness 1 (MU1) from various sources [[Hauagge and Snavely, 2012](#); [Kelman et al., 2007](#); [Sandia National Laboratories, 2016](#); [Rochester Institute of Technology, 2016](#); [Visual Geometry Group, University of Oxford, 2016](#)]



(e) Disparateness 2 (MU2) from [Razakarivony and Jurie \[2015\]](#)

Figure 6.2. Examples of different datasets: Statue of Liberty (LY), Notre Dame (ND), Half Dome in Yosemite (YO) are from [Brown et al. \[2011\]](#). Our disparate sets, MU1 and MU2, include images cropped from [Hauagge and Snavely \[2012\]](#) and others, and [Razakarivony and Jurie \[2015\]](#), respectively.

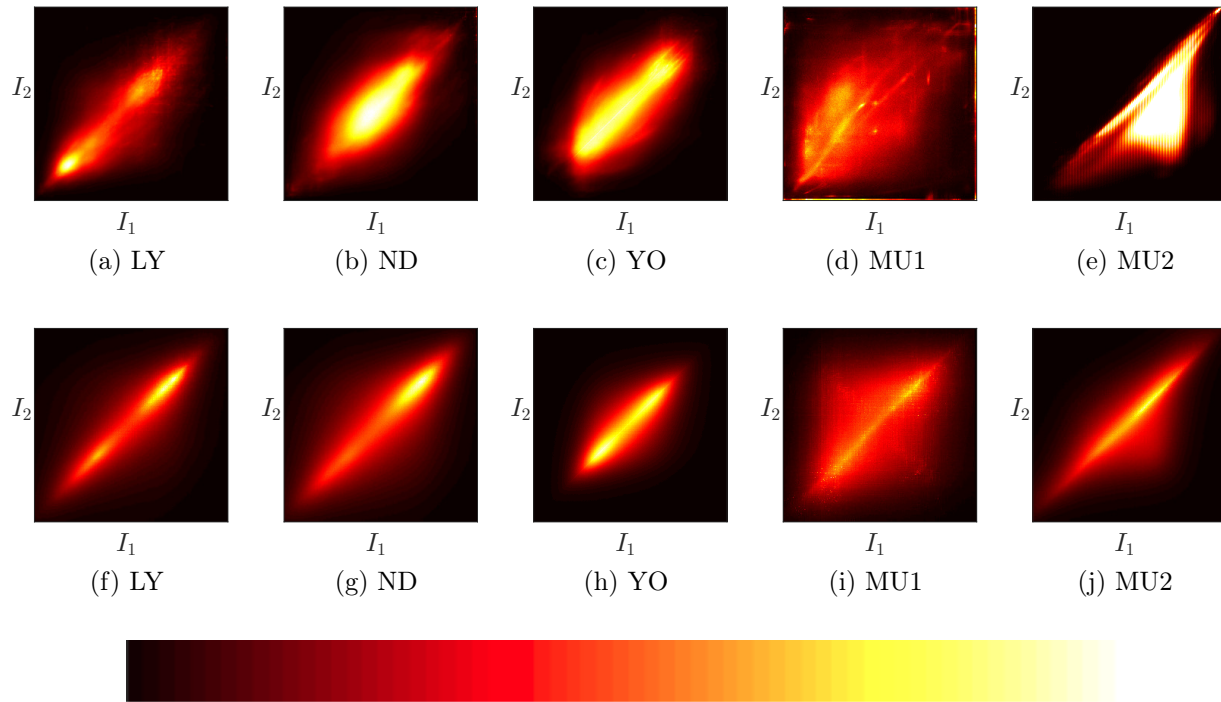


Figure 6.3. Joint distribution of corresponding pixel intensities. Hotter colors reflect more frequent occurrence. (a-e): Original patches (intensities $\in [0, 255]$). (f-j): Preprocessed patches (intensities $\in [-.3, .3]$).

pairs) and the second image pair set (980 image pairs), this resulted in around 15,000 (MU1) and 100,000 (MU2) corresponding patch pairs, respectively. Since most image descriptors consider gray-scale input, all color images are converted to gray-scale.

Note that we do not train any models from these disparate datasets (MU1 and MU2). Our goal is to learn a descriptor embedding model from the given training data (LY, ND, YO) that is effective for data of different characteristics (MU1 and MU2) as well. For disparate testing, we train a model with all three MVS subsets (LY, ND, YO).

How challenging is disparate input, and what makes it difficult? To visualize the different nature of disparate images, we conduct a simple analysis. For 10,000 random corresponding patch pairs (positive samples) in a dataset $\{\text{LY, ND, YO, MU1, MU2}\}$, we compare corresponding pixel intensities I_1 and $I_2 \in [0, 255]$ and visualize the joint distribution (I_1, I_2) in Figure 6.3 (a-e). Hotter color reflects higher occurrence. A set of pairs of identical patches will generate a distribution only on the diagonal. Compared to the standard MVS dataset (a-c), our disparate datasets (d-e) show more diluted distribution off the diagonal, meaning more challenging inputs.

Intensity mean and standard deviation normalization is a popular preprocessing approach to standardize images so that they are invariant to overall illumination and contrast changes.

We also visualize the same analysis after such preprocessing (f-j). Although input pairs correlate more strongly, the disparate inputs still show more variability in the values of corresponding pixels.

6.3 Artificial Intensity Remapping

A more robust system should work on input beyond the style(s) available in the training set. What if during learning we expose a learning process to various versions of corresponding patches? Under this reasoning, we generate randomized *artificial intensity remappings*. One can think of a mapping function $F : x \rightarrow y$ ($0 \leq x, y \leq 255$) that changes the look of a patch, i.e., changes intensity from x to y .

We first define k control points, (x_i, z_i) for $i = 1, 2, \dots, k$:

$$a \leftarrow \mathcal{B}(0, 1) \quad (\text{binary random sample}) \quad (6.1)$$

$$x_i \leftarrow 255(i - 1)/(k - 1) \quad (\text{uniform subdivision}) \quad (6.2)$$

$$r_i \leftarrow \mathcal{U}(0, 1) \quad (\text{uniform random sample}) \quad (6.3)$$

$$s_i \leftarrow (-1)^a \sum_{j=1}^i r_j \quad (\text{accumulated sum}) \quad (6.4)$$

$$z_i \leftarrow 255/(\max_i(s_i) - \min_i(s_i))(s_i - \min_i(s_i)) \quad (6.5)$$

where $\mathcal{B}(\alpha, \beta)$ denotes a uniform binary sample, either α or β , and $\mathcal{U}(\alpha, \beta)$ denotes a uniform random sample in $[\alpha, \beta)$. The s_i s are the accumulated sum of random samples (r_i s), and a determines whether the s_i s are increasing positive values or decreasing negative values. The z_i s are the normalization of the s_i s such that the minimum and maximum values are 0 and 255.

Taking (x_i, z_i) as k control points, equally spaced on the x axis, we interpolate them with a second order interpolating spline (i.e., going through each control point with continuous first derivatives). This spline defines a mapping function Z that maps a pixel intensity x to $Z(x)$. Figure 6.4a illustrates an example of randomly generated s_i s ($a = 1$, before normalization) and the spline interpolation.

To introduce noise, we also perturb the function Z at every possible integer point x_j by a random amount less than a constant perturbation parameter, p .

$$x_j \leftarrow j \quad \text{for } j = 0, 1, \dots, 255 \quad (6.6)$$

$$b_j \leftarrow \mathcal{U}(-1, 1) \quad (6.7)$$

$$y_j \leftarrow Z(x_j) + b_j \cdot p \quad (\text{perturbation}) \quad (6.8)$$

Then we re-interpolate these new (x_i, y_i) points using linear interpolation to obtain our final function F that maps a pixel intensity x to $F(x)$.

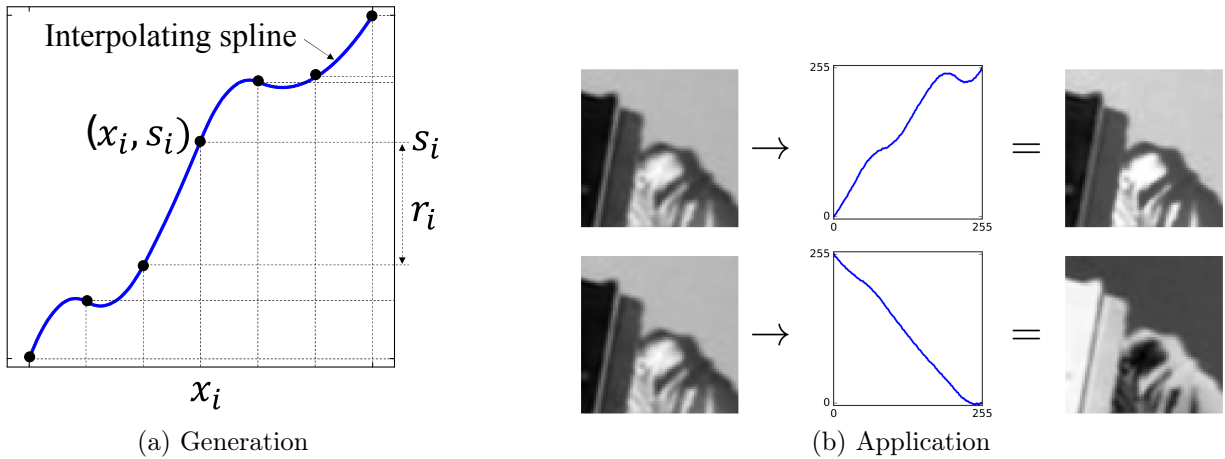


Figure 6.4. Examples of AIR generation and application.

	architecture	dimension / connectivity pooling / activation
Simo-Serra et al. [2015]	C(7,2,32)-P(2)-C(6,3,64)- P(3)-C(5,4,128)	128 / sparse L_2 / tanh
Kumar et al. [2016]	B(7,3,96)-P(2)-B(5,1,192)-P(2)- B(3,1,256)-B(1,1,256)-C(1,1,1)	256 / regular max / ReLU
AIR	C(7,2,32)-P(2)-C(6,3,64)- P(3)-C(5,4,128)	128 / regular L_2 / tanh

Table 6.1. Architectural comparison with the most closely related work: $C(w, s, n)$ denotes a convolution layer with n filters of size $w \times w$ and stride s ; $B(w, s, n)$ denotes $C(w, s, n)$ with batch normalization; $P(w)$ denotes a pooling layer of size $w \times w$ with stride w .

We show two examples of these AIR random mapping functions, together with patches before and after applying the functions in Figure 6.4b; seven variations of resulting patches were shown in Figure 6.1 ($k = 7, p = 10$). When training, we generate a different such AIR mapping function for every patch and apply it to the patch, followed by random horizontal and/or vertical flipping, and mean and standard deviation normalization.

6.4 Experiments

6.4.1 Setup

To show the efficacy of AIR, we compare our performance to [Simo-Serra et al. \[2015\]](#) and [Kumar et al. \[2016\]](#). The architectural differences of the underlying CNNs to which they are applied are summarized in Table 6.1. As key strategies, [Simo-Serra et al. \[2015\]](#) introduces a scheme of mining hard samples with sparse connectivity between layers, whereas [Kumar et al. \[2016\]](#) introduces an extra global term in their objective function.

We basically follow the architecture of [Simo-Serra et al. \[2015\]](#). Since we found that sparse connectivity is not easy to implement for some libraries (e.g. Tensorflow, which we use), we used regular connectivity despite possible disadvantages in performance. Note also that the dimensions of resulting descriptors are 128 for us and [\[Simo-Serra et al., 2015\]](#), but 256 for [\[Kumar et al., 2016\]](#). Descriptors of higher dimension may have an intrinsic advantage in description ability at the cost of higher computation.

For the objective loss function, instead of the hinge embedding loss [\[Mobahi et al., 2009\]](#) using the L_2 norm that [Simo-Serra et al. \[2015\]](#) uses:

$$l_1(\mathbf{x}_1, \mathbf{x}_2, y) = \begin{cases} \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2 & y = 1 \\ \max(0, C - \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2) & y = 0 \end{cases} \quad (6.9)$$

we use the form with squared terms as in [Hadsell et al. \[2006\]](#):

$$l_2(\mathbf{x}_1, \mathbf{x}_2, y) = \begin{cases} \frac{1}{2} (\|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2)^2 & y = 1 \\ \frac{1}{2} \max(0, C - \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2)^2 & y = 0 \end{cases} \quad (6.10)$$

where \mathbf{x}_1 and \mathbf{x}_2 are the patch pair input and y is a binary label denoting a positive or negative sample, i.e, true or false match (Figure 2.1). In both cases, the objectives are to reduce the L_2 norm for positive samples ($y = 1$), and to increase the norm up to a margin parameter C for negative samples ($y = 0$). Note that negative samples whose norms are larger than C do not increase the objective loss.

In terms of impact on learning, note that for these two functions the gradient of objective l_2 is proportional to that of l_1 (see Appendix B). Since gradient descent updates are weighted by learning rates with a decay as well, it is not clear that these proportional changes in the gradient will impact the overall learning performance.

6.4.2 Network training

Following [Simo-Serra et al. \[2015\]](#), we use Mini-batch Gradient Descent with a batch size of 128, a starting learning rate of 0.01 with a decay of 0.9 every 10,000 iterations, and a

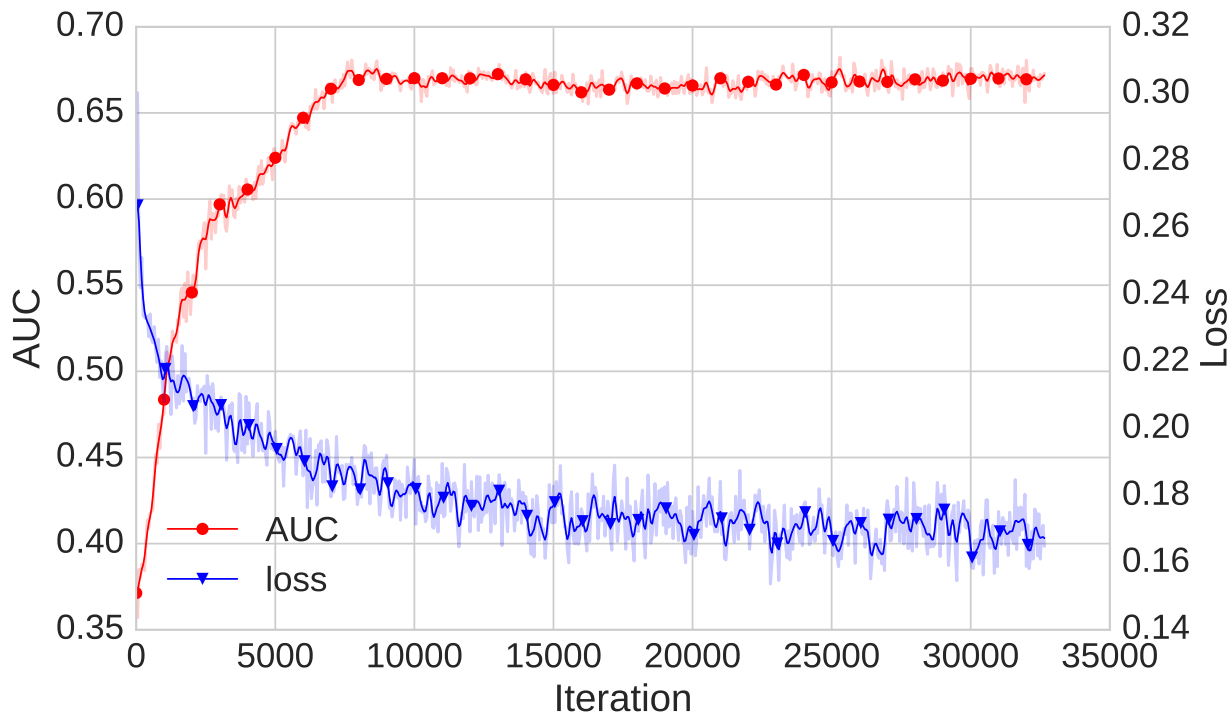


Figure 6.5. Training process of an example model training using LY and ND with a validation set from YO. The real curves are shown in light colors (light red for AUC and light blue for loss). To visualize their trends, we smooth them using exponential smoothing, and mark them in darker red and darker blue colors.

momentum of 0.9. We use a randomly selected small subset¹ of the data for validation and stop training when the evaluation metric converges. Figure 6.5 shows typical behavior during the training processes. As iterations increase, the objective losses decrease, and the evaluation metric (Area Under Curve, i.e., AUC) increases until they eventually stabilize. We terminate training when we saw no more progress, which occurred at an iteration between 28,000 and 35,000 on the input tested.

We also adopt the mining scheme of [Simo-Serra et al. \[2015\]](#) to choose “hard” samples. Specifically, at every training iteration, a set of training samples m times larger than the actual batch size b are fetched, and their descriptors calculated (requiring forward propagation). These are then pruned down to the batch size by choosing the positive samples with the $b/2$ largest descriptor distances and the negative samples with the $b/2$ smallest descriptor distances.

We found this mining scheme to be very effective. Although [Simo-Serra et al. \[2015\]](#) used a mining factor $m = 8$, we use $m = 4$ for computational speed and memory efficiency, since we did not measure much difference in terms of performance. From the view point of

¹See Section 6.4.3 for the detailed number of training and test sets.

mining, since AIR can be regarded as intentionally generating artificial “hard” samples even from “easy” samples, less mining ($m = 4$) with AIR may be as effective as mining ($m = 8$) without AIR.

In terms of computation speed, as mentioned by [Krizhevsky et al. \[2012\]](#), data augmentation processes can be implemented so as to exploit computation parallelism with learning (e.g., data augmentation can be processed on the CPU while the GPU is training with the previous batch). This means that we can obtain the benefits of AIR at almost no trade-off cost in wall-clock training time.

Since each of our training sets consists of 300,000 samples (see Section 6.2), training an example model using LY and ND (i.e., 600,000 samples) for 30,000 iterations, a batch size of 128, and a mining factor of 4 will run for $(30,000 \times 128 \times 4)/600,000 = 25.6$ epochs. In other words, we end up training with around 26 AIR-generated versions per original patch input.

6.4.3 Evaluation

Non-disparate-imagery tests We follow the evaluation method of [Simo-Serra et al. \[2015\]](#). The testing process is as follows. We randomly pick 5,000 patches. For each of the selected patches, we select one corresponding patch and 1,000 other random patches, and calculate the descriptor distances of 1 true pair and 1,000 false pairs. PR (precision-recall) curves and corresponding AUC are calculated from the $5000 \times (1000 + 1)$ descriptor distances. For validation sets, we use a small subset (100 sets of 1 true vs. 1000 false samples) chosen separately.

The PR curves of the cross-subset MVS tests (LY, ND, YO) are shown in Figure 6.6, and AUCs are shown in Table 6.2. Numbers and letters in parentheses represent descriptor dimension and type (f:floating point, b:binary), respectively. In the LY and ND tests, our model trained with AIR had the best performance. In YO test, our model shows almost the same performance as the best performance among state-of-the-art algorithms. On average, our model shows the best performance.

In regards to performance reporting, there are three things to note: (1) we reconstructed the MVS training and test sets following the randomized procedure of [Simo-Serra et al. \[2015\]](#), therefore our test sets are not exactly the same as theirs. In Table 6.2, Global [[Kumar et al., 2016](#)] and AIR are tested on our reconstructed MVS sets; we quote the other results from [Simo-Serra et al. \[2015\]](#) since we found that CNN [[Simo-Serra et al., 2015](#)] performed almost identically when run on our reconstructed test sets. (2) Because Global [[Kumar et al., 2016](#)] provides only LY-trained, ND-trained, and YO-trained models, we report whichever gives the best performance in Tables 6.2 and 6.3. (3) Because SIFT is a hand-crafted descriptor, the “train” columns in Table 6.2 and 6.3 are not applicable.

Disparate-imagery tests For the disparate tests (MU1, MU2), we compare our model trained with AIR to state-of-art descriptors [[Simo-Serra et al., 2015](#); [Kumar et al., 2016](#)] as well as SIFT. We also measured the performance of our model without AIR to isolate the

train	test	SIFT [44] (128f)	BGM [78] (256b)	L-BGM [78] (64f)	BinBoost [78] (256b)	VGG [69] (80f)	CNN [68] (128f)	Global [36] (256f)	AIR (128f)
ND,YO	LY	0.226	0.268	0.355	0.410	0.558	0.608	0.572	0.646
LY,YO	ND	0.349	0.487	0.495	0.549	0.663	0.667	0.633	0.712
LY,ND	YO	0.425	0.495	0.517	0.533	0.709	0.545	0.590	0.705
average		0.333	0.417	0.456	0.497	0.643	0.607	0.598	0.688

Table 6.2. AUC of precision-recall curves for cross-subset MVS test (bold denotes top performance).

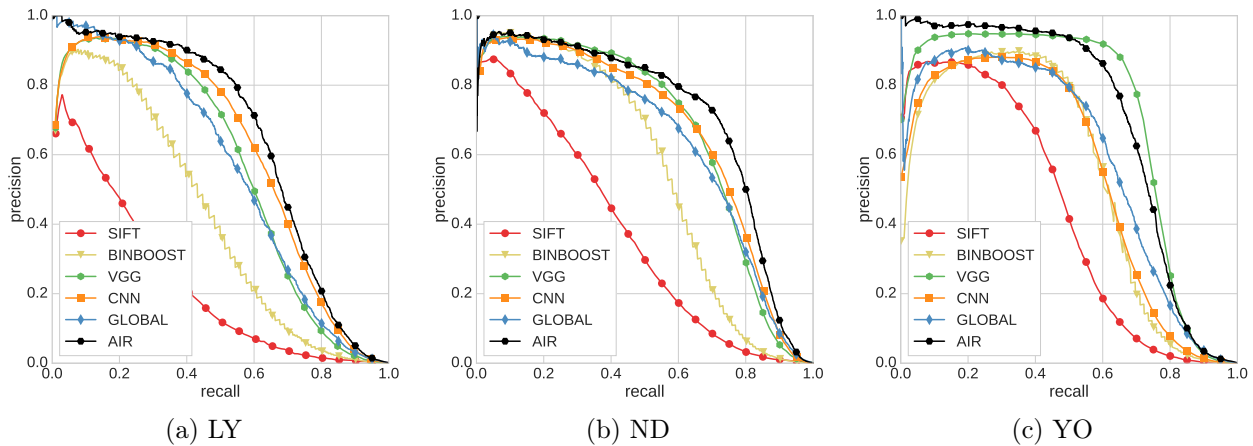


Figure 6.6. Precision-recall curves for each non-disparate test set.

train	test	SIFT [44] (128f)	CNN [68] (128f)	Global [36] (256f)	w/o AIR (128f)	w/ AIR (128f)
LY,ND,YO	MU1	0.216	0.217	0.211	0.301	0.316
LY,ND,YO	MU2	0.437	0.633	0.681	0.660	0.735

Table 6.3. AUC of precision-recall curves for disparate test (bold denotes top performance).

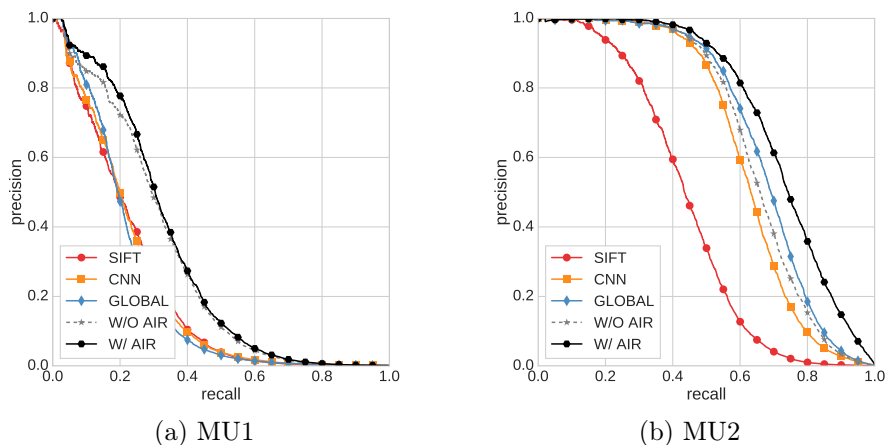


Figure 6.7. Precision-recall curves for each disparate test set.

effect of AIR. Performance is measured with PR curves in Figure 6.7 and AUCs in Table 6.3. As seen in Figure 1.2 and Figure 6.3, MU1 contains pairs with more severe appearance changes than other test sets, which explains why overall performance is lower on MU1, no matter the descriptor. Deep-learning-based descriptors trained on the MVS dataset will have not seen such input during training. In this case, a hand-crafted descriptor, SIFT, engineered by human reasoning, works similarly to or even better than [Simo-Serra et al. \[2015\]](#) and [Kumar et al. \[2016\]](#). Despite the fact that we set up most of our network architecture and hyper parameters similarly to those of [Simo-Serra et al. \[2015\]](#), even our model without AIR outperforms it. Note again that the choice of the hyper parameter C (we set our own value $C = 0.9$ since it was not specified for theirs) and the square terms differ from [Simo-Serra et al. \[2015\]](#). The mining factor ($m = 4$) and the regular connectivity that we used are the ones they reported as suboptimal. In regards to disparate performance reporting, note that all algorithms are tested using identical test sets, and we use [Vedaldi and Fulkerson \[2010\]](#) for SIFT implementation.

Although our base model (w/o AIR) brought a surprising jump in performance, one can still see the additional benefit of AIR. The performance of the other algorithms are all similar to each other (AUC differing by 0.001–0.006) for the most challenging input of MU1. The performance gain of 0.015 using AIR is a 5% gain in AUC compared to our same implementation without AIR. In the case of the MU2 test, the performance of our base model is similar to that of CNN, and inferior to that of Global. However, AIR delivers an 11% boost in performance, and attains the best results.

6.5 Comparison with DUDE

So far we have evaluated the performance of AIR based on “patch” tests, where we collected numerous examples of true and false matches between small sub-patches within dataset.

		detectors	
		GRID	SIFT
descriptors	DUDE	0.63	0.35
	AIR	0.52	0.34

Table 6.4. Mean average precision (mAP) evaluation and comparison between AIR-augmented deep-learning descriptor and DUDE. DUDE is comparable to AIR-augmented deep-learning descriptor when using SIFT as the detector, and a bit better in the case of a hypothetical perfect detector as represented by GRID.

In addition, as in Chapter 4, we conducted “practical” tests where features (keypoints) are extracted using two detector systems, GRID and SIFT (see Chapter 4), and descriptors are computed for the extracted features. For image pairs, we used the 46 images of the disparate architecture dataset [Hauagge and Snavely, 2012] (a few images are illustrated in Figure 1.2d). Table 6.4 summarizes the results. Although our deep-learning-based descriptor with AIR showed overall better performance than the other comparable deep-learning descriptors, it showed similar or a bit lower performance than DUDE’s. Very recently, Schönberger et al. [2017] compared hand-engineered features versus data-driven features, and also reported that advanced hand-engineered features still perform on par or better than data-driven features. It is interesting that, unlike higher-level tasks such as object classification/recognition and image segmentation, data-driven approaches do not obviously outperform sophisticated engineered approaches in image feature research.

We hypothesize that data-driven image descriptors could be much more powerful if trained directly on disparate imagery datasets. Since (1) data-driven approaches may have weaknesses due to limited training data and over-fitting and (2) our “disparate” cases should cover a large range of possible appearance changes, we expect that an effort to collect a much larger dataset of disparate images would be very valuable to fill this gap.

6.6 Summary

We present a novel data augmentation scheme, Artificial Intensity Remapping (AIR). AIR artificially generates adversarial but still realistic input to help learning for better generalization. To show the efficacy of AIR, we build disparate test beds from various sources. This testing on disparate appearance input demonstrates the benefits of AIR in addressing the fact that successful performance in deep learning, with its huge number of parameters, may result from overfitting to a given dataset. Various experiments show that our model trained using only non-disparate data with AIR outperforms state-of-the-art algorithms not only for non-disparate data (intra-dataset cross tests) but also for disparate data (adversarial inter-dataset cross tests).

Chapter 7

Conclusion and Future Work

In this dissertation, we hypothesized that, in order to find correspondences across image pairs with dramatically inconsistent appearance, it is advantageous to make use of information that captures aspects of shape that could be preserved despite significant appearance changes (e.g., reversal of brightness and darkness, partial absence of edges, etc.). For this purpose, we explored capturing the distributions of line segments. Even though, by relying on line segments, our algorithm is free from the effects of color, intensity, and/or texture changes, since dramatic appearance changes are highly likely to affect the detection of line segments as well, we wanted our algorithm to be robust against such changes in detection. We also aimed for a computationally efficient algorithm.

As a first step, we demonstrated the effectiveness of using (the distribution of) line segments as cues in Chapter 3 for finding correspondences with the RANSAC algorithm. The result shows one can utilize the distribution of line segments to register challenging input for which existing methods fail. However, relying on the naïve RANSAC algorithm without appropriate descriptors requires a huge number of trials to be eventually successful. Therefore, in Chapter 4, we consider how can we capture the distributions of line segments and introduce a descriptor system, called DUDE, which is designed for line segments, and consistent even when the input distributions consist of disconnected line segments and/or their unstable endpoints. The experimental results show that DUDE descriptors significantly outperform most existing descriptors. In addition, DUDE shows equivalent performance to the state-of-the-art with significantly less computational cost. In Chapter 5, beyond image matching, we explored a new application of DUDE where we utilize the DUDE descriptor to improve the performance of a new approach to autograding multiview engineering drawings that we introduce. The system can compare two multiview drawings, a reference solution and a student's submission, that may have inconsistent translations, scales, mirroring, skew, and/or rotation; all of which must be distinguished from allowable differences in scale, offset, and translation to reliably identify and classify errors in the students' drawings. By using DUDE, we improve computational speed significantly.

Admittedly, DUDE has limitations. First, DUDE requires setting several parameters (including parameters related to line segment detection). A possible avenue for future re-

search is studying a systematic way of determining those parameter. Second, DUDE requires somewhat larger feature (region) than most other descriptors, so that feature regions include a number of line segments. However, this can also be an advantage in that DUDE can be used in a complimentary manner, alongside existing descriptors. Third, DUDE may be more effective with images including more linear features than curvy features, which are detected as short line segments.

The recent focus of image feature extraction research (whether feature detection or description) research has turned towards deep-learning based features, inspiring our work introducing AIR as an potential aggressive data augmentation in Chapter 6. However, the deep-learning-based approaches have weakness in the need for extensive training data. We show that deep-learning-based descriptors, even with the help of AIR, are not yet as effective as our handcrafted DUDE descriptor when applied to disparate images. A promising direction for future work is exploring how one could combine the advantages of handcrafted descriptors and deep-learning-based descriptors.

Appendix A

Histogram comparison

There are many possible measures to compare two histograms. Here we introduce a few of popular measures. For given d -dimension histograms P and Q :

$$\text{Euclidean } L_2 \quad \sqrt{\sum_{i=1}^d (P_i - Q_i)^2} \quad (\text{A.1})$$

$$\text{Minkowski } L_p \quad \sqrt[p]{\sum_{i=1}^d (P_i - Q_i)^p} \quad (\text{A.2})$$

$$\text{Intersection} \quad \sum_{i=1}^d \min(P_i, Q_i) \quad (\text{A.3})$$

$$\text{Harmonic mean} \quad 2 \sum_{i=1}^d \frac{P_i Q_i}{P_i + Q_i} \quad (\text{A.4})$$

$$\text{Bhattacharyya} \quad -\ln \sum_{i=1}^d \sqrt{P_i Q_i} \quad (\text{A.5})$$

$$\chi^2 \quad \sum_{i=1}^d \frac{(P_i - Q_i)^2}{P_i + Q_i}. \quad (\text{A.6})$$

See [Cha \[2007\]](#) for more metrics and detailed comparisons.

Appendix B

Objective loss functions

The gradients using the two variants of hinge embedding loss are proportional as follows. One can show that when $y = 1$,

$$\frac{\partial l_1(\mathbf{x}_1, \mathbf{x}_2, y)}{\partial W} = \frac{\partial \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2}{\partial W} \quad (\text{B.1})$$

$$\begin{aligned} \frac{\partial l_2(\mathbf{x}_1, \mathbf{x}_2, y)}{\partial W} &= \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2 \frac{\partial \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2}{\partial W} \\ &= \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2 \frac{\partial l_1(\mathbf{x}_1, \mathbf{x}_2, y)}{\partial W} \end{aligned} \quad (\text{B.2})$$

and when $y = 0$ and $\|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2 < C$,

$$\frac{\partial l_1(\mathbf{x}_1, \mathbf{x}_2, y)}{\partial W} = -\frac{\partial \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2}{\partial W} \quad (\text{B.3})$$

$$\begin{aligned} \frac{\partial l_2(\mathbf{x}_1, \mathbf{x}_2, y)}{\partial W} &= -(C - \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2) \frac{\partial \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2}{\partial W} \\ &= (C - \|D(\mathbf{x}_1) - D(\mathbf{x}_2)\|_2) \frac{\partial l_1(\mathbf{x}_1, \mathbf{x}_2, y)}{\partial W}. \end{aligned} \quad (\text{B.4})$$

Bibliography

- Achanta, R., Shaji, A., and Smith, K. (2012). SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *TPAMI*, 34(11):2274–2282.
- AUTODESK (2000). AutoCAD Drawing Compare Plug-in. <https://apps.exchange.autodesk.com/VLTC/en/Detail/Index?id=appstore.exchange.autodesk.com%3Adrawizgcompare%3Aen>. Last accessed on Oct 20, 2014.
- Balntas, V., Riba, E., Ponsa, D., and Mikolajczyk, K. (2016). Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC*, volume 33, page 2011.
- Bansal, M. and Daniilidis, K. (2013). Joint spectral correspondence for disparate image matching. In *Computer Vision and Pattern Recognition*, pages 2802–2809.
- Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522.
- Bertoline, G., Wiebe, E., Hartman, N., and Ross, W. (2002). *Technical Graphics Communication*. McGraw-Hill Science/Engineering/Math.
- Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., Lecun, Y., Moore, C., Säckinger, E., and Shah, R. (1993). Signature Verification Using a "Siamese" Time Delay Neural Network. *International Journal of Pattern Recognition and Artificial Intelligence*, 07(04):669–688.
- Brown, M., Hua, G., and Winder, S. (2011). Discriminative Learning of Local image Descriptors. *Transactions on Pattern Analysis and Machine Intelligence*, 33(1):43–57.
- Brown, M. and Lowe, D. (2003). Recognising panoramas. *International Conference on Computer Vision*.
- Brown, M. and Lowe, D. G. (2005). Unsupervised 3D object recognition and reconstruction in unordered datasets. In *Fifth International Conference on 3-D Digital Imaging and Modeling*, pages 56–63.

- Canny, J. (1986). A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698.
- Cao, Y. (Last accessed on Apr 14, 2015). Hungarian algorithm for linear assignment problems (v2.3). <http://www.mathworks.com/matlabcentral/fileexchange/20652>.
- Cha, S.-h. (2007). Comprehensive Survey on Distance / Similarity Measures between Probability Density Functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4):300–307.
- Choi, S., Kim, T., and Yu, W. (1997). Performance evaluation of RANSAC family. *Journal of Computer Vision*.
- Coiras, E., Santamarı, J., and Miravet, C. (2000). Segment-based registration technique for visual-infrared images. *Optical Engineering*, 39(1):282–289.
- Dubrofsky, E. and Woodham, R. J. (2008). Combining Line and Point Correspondences for Homography Estimation. In *Advances in Visual Computing*, pages 202–213. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Fedorov, D. V., Fonseca, L. M., Kenney, C., and Manjunath, B. S. (2003a). Automatic registration and mosaicking system for remotely sensed imagery. pages 444–451.
- Fedorov, D. V., Kenney, C., Manjunath, B. S., and Fonseca, L. M. (2003b). Image Registration With Fit Assessment Demo. <http://vision.ece.ucsb.edu/registration/imreg/>. Last accessed on May 14, 2014.
- Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.
- Geng, W., Wang, J., and Zhang, Y. (2002). Embedding visual cognition in 3D reconstruction from multi-view engineering drawings. *Computer-Aided Design*.
- Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. In *International Conference on Machine Learning*, volume 28, pages 1319–1327.
- Gradescope (2017). gradescope.com. <https://gradescope.com>. Last accessed on Apr 4, 2017.
- Graham, B. (2014). Fractional Max-Pooling. *arXiv*, pages 1–10.
- Grompone von Gioi, R., Jakubowicz, J., Morel, J.-M., and Randall, G. (2012). Lsd: a line segment detector. *Image Processing On Line*, 2:35–55.

- Habib, A. F. and Alruzouq, R. I. (2004). Line-based modified iterated Hough transform for automatic registration of multi-source imagery. *The Photogrammetric Record*, 19(105):5–21.
- Hadsell, R., Chopra, S., and Lecun, Y. (2006). Dimensionality Reduction by Learning an Invariant Mapping. In *Conference on Computer Vision and Pattern Recognition*.
- Harris, C. and Stephens, M. (1988). A combined corner and edge detector. *Proceedings of the Alvey Vision Conference*, pages 147–151.
- Hartley, R. and Zisserman, A. (2003). Multiple View Geometry in Computer Vision.
- Hauagege, D. C. and Snavely, N. (2012). Image matching using local symmetry features. In *Computer Vision and Pattern Recognition*, pages 206–213.
- Irani, M. and Anandan, P. (1998). Robust multi-sensor image alignment. *International Conference on Computer Vision*, pages 1–19.
- Kelman, A., Sofka, M., and Stewart, C. V. (2007). Keypoint descriptors for matching across multiple image modalities and non-linear intensity variations. *Computer Vision and Pattern Recognition*, pages 1–7.
- Kim, H., Correa, C., and Max, N. (2014a). Automatic Registration of LiDAR and Optical Imagery using Depth Map Stereo. *Computational Photography, IEEE International Conference on*, pages 205–212.
- Kim, H., Thiagarajan, J., and Bremer, P.-T. (2014b). Image segmentation using consensus from hierarchical segmentation ensembles. In *IEEE International Conference on Image Processing*, pages 3272–3276.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, pages 1–9.
- Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:87–97.
- Kumar, B. G. V., Carneiro, G., and Reid, I. (2016). Learning Local Image Descriptors with Deep Siamese and Triplet Convolutional Networks by Minimising Global Loss Functions. In *Conference on Computer Vision and Pattern Recognition*, pages 5385–5394.
- Kwon, Y. P. (2014). Line segment-based aerial image registration. Master’s thesis, EECS Department, University of California, Berkeley.
- Kwon, Y. P. and McMains, S. (2015). An Automated Grading / Feedback System for 3-View Engineering Drawings using RANSAC. In *ACM Conference on Learning at Scale*, pages 157–166, Vancouver, BC, Canada. ACM.

- Lee, H. and Han, S. (2005). Reconstruction of 3D interacting solids of revolution from 2D orthographic views. *Computer-Aided Design*, 37(13):1388–1398.
- Lieu, D. and Sorby, S. (2009). *Visualization, Modeling, and Graphics for Engineering Design*. Delmar Learning, 3rd edition.
- Lin, K., Lu, J., Chen, C.-S., and Zhou, J. (2016). Learning Compact Binary Descriptors with Unsupervised Deep Neural Networks. In *Conference on Computer Vision and Pattern Recognition*.
- Lin, M., Chen, Q., and Yan, S. (2013). Network In Network. *arXiv*, page 10.
- Lindeberg, T. (1998). Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):79–116.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.
- Matas, J., Chum, O., Urban, M., and Pajdla, T. (2002). Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference*, pages 384–393.
- Mathworks (2000a). MATLAB CENTRAL, File Exchange. <http://www.mathworks.com/matlabcentral/fileexchange>. Last accessed on Oct 20, 2014.
- Mathworks (2000b). Read DXF File Data. <http://www.mathworks.com/matlabcentral/fileexchange/24572-read-dxf-file-data>. Last accessed on Oct 20, 2014.
- Mikolajczyk, K. and Schmid, C. (2004). Scale & affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86.
- Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630.
- Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, a., Matas, J., Schaffalitzky, F., Kadir, T., and Van Gool, L. (2005). A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1-2):43–72.
- Miriana (2017). Technical Drawing for Mechanical Engineering. <https://miriadax.net/web/technical-drawing-for-mechanical-engineering-dibujo-tecnico-para-ingenieria-mecanica>. Last accessed on Apr 4, 2017.
- MIT Open Courseware (2017). Design Handbook: Engineering Drawing and Sketching. https://ocw.mit.edu/courses/mechanical-engineering/2-007-design-and-manufacturing-i-spring-2009/related-resources/drawing_and_sketching/. Last accessed on Apr 4, 2017.

- Mobahi, H., Collobert, R., and Weston, J. (2009). Deep Learning from Temporal Coherence in Video. In *International Conference on Machine Learning*, pages 737–744.
- Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38.
- Nistér, D. (2005). Preemptive RANSAC for live structure and motion estimation. *Machine Vision and Applications*.
- Qi, Y., Song, Y.-Z., Zhang, H., and Liu, J. (2016). Sketch-based Image Retrieval via Siamese Convolutional Neural Network. In *International Conference on Image Processing*, number iii, pages 2460–2464.
- Raguram, R., Frahm, J.-M., and Pollefeys, M. (2008). A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus. In *ECCV 2008*, pages 500–513.
- Razakarivony, S. and Jurie, F. (2015). Vehicle Detection in Aerial Imagery : A Small Target Detection Benchmark. *Journal of Visual Communication and Image Representation*, 34:187–203.
- Rochester Institute of Technology (Last accessed on Nov 11, 2016). 3D-Rochester: Image and LiDAR Dataset. <http://dirsapps.cis.rit.edu/3d-rochester/data.html>.
- Sandia National Laboratories (Last accessed on Nov 11, 2016). SAR images from Sandia National Laboratories. https://web.archive.org/web/*/http://www.sandia.gov/radar/images/*.
- Schmid, C. and Zisserman, A. (1997a). Automatic line matching across views. In *Computer Vision and Pattern Recognition*, pages 666–671.
- Schmid, C. and Zisserman, A. (1997b). Automatic line matching across views. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 666–671.
- Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum*.
- Schönberger, J. L., Hardmeier, H., Sattler, T., and Pollefeys, M. (2017). Comparative Evaluation of Hand-Crafted and Learned Local Features. *CVPR*, pages 1482–1491.
- Sharvit, D., Chan, J., Tek, H., and Kimia, B. B. (1998). Symmetry-based indexing of image databases. *Journal of Visual Communication and Image Representation*, 9(4):366–380.
- Shechtman, E. and Irani, M. (2007). Matching local self-similarities across images and videos. *Computer Vision and Pattern Recognition*.

- Shin, B. S. and Shin, Y. G. (1998). Fast 3d solid model reconstruction from orthographic views. *Computer-Aided Design*, 30(1):63–76.
- Simo-Serra, E., Trulls, E., Ferraz, L., Kokkinos, I., Fua, P., and Moreno-Noguer, F. (2015). Discriminative Learning of Deep Convolutional Feature Point Descriptors. In *International Conference on Computer Vision*, pages 118–126. IEEE.
- Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Learning Local Feature Descriptors Using Convex Optimisation. *Transactions on Pattern Analysis and Machine Intelligence*, pages 1–14.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. In *International Conference on Learning Representations*, pages 1–14.
- Stewart, C. V., Tsai, C.-L., and Roysam, B. (2003). The dual-bootstrap iterative closest point algorithm with application to retinal image registration. *IEEE transactions on medical imaging*, 22(11):1379–1394.
- Suh, Y. S. and McCasland, J. (2009). Interactive Construction of Solids from Orthographic Multiviews for an Educational Software Tool. *Computer-Aided Design and Applications*, 6(2):219–229.
- Taigman, Y., Yang, M., Ranzato, M. A., and Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *Conference on Computer Vision and Pattern Recognition*.
- Tensorflow™ (Last accessed on Nov 11, 2016). Tensorflow. <https://www.tensorflow.org>.
- Tola, E., Lepetit, V., Fua, P., and Member, S. (2010). Daisy: an efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830.
- Tombari, F., Franchi, A., and Di, L. (2013). Bold features to detect texture-less objects. *International Conference on Computer Vision*, pages 1265–1272.
- Torr, P. H. S. and Zisserman, A. (2000). MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156.
- Trzcinski, T., Christoudias, M., Fua, P., and Lepetit, V. (2013). Boosting Binary Image Descriptors. In *Conference on Computer Vision and Pattern Recognition*, pages 2874–2881.
- Van De Sande, K., Gevers, T., and Snoek, C. (2010). Evaluating color descriptors for object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1582–1596.

- Vedaldi, A. and Fulkerson, B. (2010). Vlfeat - an open and portable library of computer vision algorithms. *International Conference on Multimedia*, 3(1):1–4.
- Visual Geometry Group, University of Oxford (Last accessed on Nov 11, 2016). Affine Covariant Regions Datasets. <http://www.robots.ox.ac.uk/~vgg/data/data-aff.html>.
- Wang, L., Neumann, U., and You, S. (2009a). Wide-baseline image matching using line signatures. In *International Conference on Computer Vision*, pages 1311–1318. IEEE.
- Wang, W. and Grinstein, G. G. (1993). A survey of 3d solid reconstruction from 2d projection line drawings. *Computer Graphics Forum*, 12(2):137–158.
- Wang, Z., Fan, B., and Wu, F. (2011). Local intensity order pattern for feature description. *International Conference on Computer Vision*, pages 603–610.
- Wang, Z. and Latif, M. (2003). Reconstruction of a 3D solid model from orthographic projections. In *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, pages 75–82.
- Wang, Z., Liu, H., and Wu, F. (2009b). Msld: A robust descriptor for line matching. *IEEE International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics*, 42:128–133.
- Yi, K. M., Trulls, E., Lepetit, V., and Fua, P. (2016). LIFT: Learned invariant feature transform. *ECCV*, 9910 LNCS:467–483.
- Zagoruyko, S. and Komodakis, N. (2015). Learning to compare image patches via convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, pages 4353–4361.
- Zitová, B. and Flusser, J. (2003). Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000.