Assistive Visual Tools For Surgery

Austin Reiter

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2013

ABSTRACT

Assistive Visual Tools For Surgery

Austin Reiter

This thesis describes efforts towards the development of computational vision tools to assist physicians during surgical procedures. Surgical robotics has come a long way towards improving patient care and enhancing the abilities of the surgeon. It was originally introduced to address the limitations of manual laparoscopy, whereby long rigid tools are being used to perform complicated surgical procedures. However, as we continue to reduce the invasiveness of this hardware and the tools becomes more complicated (e.g., flexible, articulated), intelligent algorithms are necessary to assist the surgeon in both controlling the surgical robot and making it smarter.

Tool tracking algorithms are investigated to extract the locations of the surgical tools in both 2-dimensions (for visual servoing of a motorized camera system) and 3-dimensions (for situational awareness, increased patient safety, and virtual measuring capabilities). This work is shown in real surgical scenarios, such as manual human laparoscopies as well as robotic procedures using Intuitive Surgical's da Vinci® robot.

Additionally, newer procedural paradigms which reduce the invasiveness of surgical access (e.g., Single-Port Access (SPA), Natural Orifice Translumenal Endoscopic Surgery (NOTES)) require more flexible and dexterous hardware. Vision is used to measure the state of continuum ("snake") robots in order to provide feedback to control systems and be able to perform fully-automated, closed-loop actions (e.g., suturing, biopsies, etc). This work is performed on the Insertable Robotic Effector Platform (IREP), which is a dexterous, 2-arm snake manipulator with a motorized stereo camera system designed for SPA/NOTES procedures.

Table of Contents

List of Figures

List of Tables

Acknowledgments

When I picked up my comfortable life in Boston to follow my wife-to-be blindly into New York City, not in a million years did I envision myself sitting where I am today. I was employed at a comfortable job, working with fantastic people, and my wife got her *residency match* in New York City. Although there was no doubt in my mind I was going to follow her, some emotions ensued, conversations took place, and I traveled four hours south down I-95 with everything I own in the back of an SUV without a plan in the world of what to do next.

At the time, New York City wasn't the most tech-friendly place to land an interesting job. I searched, emailed, and emailed some more, and sure enough, a Professor at Columbia University agreed to talk with me about graduate opportunities in the Computer Science department. He showed me around his lab, asked me a few questions, and several months later I was admitted into the Ph.D. program at Columbia under the advisement of Dr. Peter K. Allen.

When I think back to what got me here, I realized one important thing: cherish the relationships that are presented to you during life, because you never know how each and every step along the way will shape what you become in the future. When I began my journey at BAE Systems working in Computer Vision, I wasn't initially offered a job. Apparently there was some trepidation on their part about my ability to excel in their prestigious computer vision group. I had another job offer on the table from somewhere else, and they decided to give me a shot. I can easily say today that without that moment, I certainly am not writing my Ph.D. thesis. The experience I gathered from the people I worked with there was what put me through and enabled me to succeed in a Ph.D. research environment. But I'm jumping ahead a bit.

I can't possibly thank a single person before I thank my beautiful and supporting wife Dena. She pushes me all the time to do what makes me happy and make sure I'm enjoying my life. She listens to my technical babble constantly, pretending to be interested, just to entertain me. She makes me feel proud of what I do, and going home to her every day is the best thing I could have done in life. And so for that and all the positive reinforcement I'm eternally grateful.

Next I must thank my mom, Marilyn, and my step-dad, Rick. Like many parents, they think there are no limits to what I'm capable of, past the point of embarrassment. They give lots of compassion and support to me, and have helped me in very important ways to both get through this challenging accomplishment as well as generally through the tough times in life. My mom has afforded me opportunities in life that a lot of people are not fortunate enough to have, and if she doesn't know how thankful I am of that then reading this should leave no confusion on that matter.

My sister Robyn and I have a rare relationship. When many people see how well we get along, they are often confused: how can an older sister and a younger brother not hate each other and fight all the time? Well with us it's different. I think we help each other get through life together, and she had an important part in raising me growing up, despite our four year age difference. We have a lot of fun together and she, too, has shaped my personality enormously. John Cooney helps in indescribable ways by keeping Robyn happy, making life for all of us more enjoyable!

Doctoral degrees don't happen alone. They involve numerous people helping out in different ways. Besides the family support at home, one requires a huge amount of professional support. The leader of this effort is the Ph.D. advisor. When I tried to envision what a Ph.D. advisor was going to be like when first starting out, Peter was not what I had in my mind. A graduate advisor has a harder job than a typical boss at a company. He or she must guide, teach, and prepare all at the same time. Somehow during all this Peter found the time to be a friend too. I can come in on a Monday morning, talk about Sunday's NFL games, ask about his Fire Island house, and jump right into how to improve my IROS paper to get accepted, all within an hour. When Peter attended my wedding many of my friends were asking who that guy was who was getting down so hard to the music. Peter gave me a long string to develop my research focus, and in so doing I think we developed a great mutual respect for each other that I'm sure will continue well past my exit from Columbia.

During my Ph.D., I developed great friendships with Hao Dang, Jon Weisz, Thomas Berg, Wei-Chun Yuan, Konstantinos Iliopoulos, Long Wang, and Sonny Hu. Hao helped me talk through a lot of my research problems, attempted to teach me a little Chinese (to no avail), and struggled through my inabilities to ever choose a place for lunch. Jon similarly got me through many of the core classes, as I wondered how a human being can enjoy OS as much as he does. Jon's the guy who can answer any Linux question you can come up with, and is invaluable to have around a work place. My lunches with Tom, Slyfox, Long, Sonny, and Kostas were lots of fun, regardless of how sick I got of all the sandwich places on Amsterdam Avenue and

thwarted my constant efforts to distract them while they were trying to accomplish real work.

When I first started in the Robotics Lab, Matei Ciocarlie and Corey Goldfeder really helped make me feel welcome, showed me the ropes, and gave me the dos and don'ts in order to succeed in this lifestyle. I am very grateful to all of the help and support they gave starting out.

I did two summer internships at Intuitive Surgical, which ended up saving my thesis work. When I needed some much needed help with funding to continue my tool tracking work, Tao Zhao and Simon DiMaio from Intuitive were able to work out a deal for us to continue our work. I had two great summers with Tao. He's one of the more talented Computer Vision people I've come across, and we had lots of fun over lunch just talking about the computer vision world. My work there was both fun and productive, and I couldn't be more happy to have gone through those experiences.

As I mentioned earlier, I learned most of the software and research skills I needed to succeed at Columbia during my time at BAE Systems. I continued a work relationship with many of the folks there, such as Mark Stevens, Joel Douglas, Gil Ettinger, Mike Braun, and Yuli Friedman. At BAE Systems, Matt Antone and Jeff Wong collectively taught me more about vision (Matt) and C++ (Jeff) than I probably learned in 4 years of my undergraduate work at BU. Diane Theriault shared a wall with me and hopefully enjoyed lunches and work with me as much as I enjoyed listening to her singing Green Day out loud (thinking nobody could hear her). Victor Tom, Luis Galup, Chris Stauffer, Lori Vinciguerra, James Coggins, Steve DelMarco, and Helen Webb were also great collaborators and mentors as well as good friends.

Somebody gave me some really good advice before I started this adventure. He said to make sure to stop and appreciate my time here in the Ph.D. program as much as I can along the way because it's quite possibly the only time in my life I can work on interesting problems that are both fun and rewarding, worry-free, and that the environment I work in is going to be hard to replace once it's over. His advice could not have been more true, and I wouldn't trade this experience for anything.

Chapter 1

Introduction

1.1 Robotic Surgery

When the average person thinks of surgery, a scene is usually envisioned which is gruesome, crowded, and complicated. The reality is that surgery has come a long way since the introduction of minimally-invasive techniques in the early 1900s. Advancements in technology have revolutionized surgical tools and endoscopic imaging devices and have allowed the surgeon to perform operations faster, easier, cheaper, and safer. These improvements have come about through efforts in mechanical enhancements and smarter algorithms, as we are learning how the computer can advance the surgical field forward.

When evaluating the current state of surgery, we must concentrate on both the evolution of surgical technology along with the instruments which are used to complete the surgical tasks. It is not enough to just save the life of a sick person if their quality of life is left compromised following a surgical procedure. The pain and discomfort resulting from surgery is more frequently due to the trauma involved in gaining access to the surgical site rather than the procedure itself [Mack, 2001].

Before minimally-invasive surgery (MIS), open surgery was the approach all surgeons used to heal patients, which left large cuts and scars and left patients prone to infections by exposing large portions of their insides to potentially diseased environments. Then, in 1910 the first human laparoscopic procedure was performed by Hans Christian Jacobaeus [Hatzinger *et al.*, 2006], which began a revolution in a less-invasive and safer surgical approach. The introduction of the charge-coupled device (CCD) image sensor (1969 at AT&T Bell Labs) further pushed the field forward and made it possible to project a magnified view of the surgical environment onto a computer monitor, allowing the surgeon's free hands to operate.

With this paradigm, surgeons were able to access the surgical site using small incisions with modified tools to operate on the patient and the post-operative trauma was significantly reduced. To exemplify the effect of laparoscopy in the modern era, the most common laparoscopic technique is the cholecystectomy (the surgical removal of the gallbladder), which is performed over one million times in the U.S., 96% of which are currently performed laparoscopically [Mack, 2001].

However, laparoscopic procedures are limiting in that the surgeon is using long sticks through small incisions to perform complicated tasks. Three-dimensional vision is lost as a single camera is used to image through a small hole. Furthermore, external ergonomics are challenging as instruments must be worked "backwards", moving right to force a left motion, and vice versa. All this being said, MIS has become a standard operating choice for surgeons in recent years due to its low risks and high rewards. Patients and doctors alike prefer fewer scars and smaller incisions with quicker healing times. Currently, MIS requires multiple incisions and multiple people in the operating room, all trying to work together, but with much effort. The learning curve for surgeons to learn laparoscopic techniques is quite high, and so new technology was needed to overcome these limitations.

Despite the introduction of minimally-invasive techniques to general surgery more than 20 years ago, many of the more advanced surgical procedures are still too difficult to perform laparoscopically. Limited dexterity and reduced perception have limited surgeon's abilities to achieve comparable results to open surgery. The introduction of robotics to surgery was originally intended to enable remote surgery, whereby a physician can perform a surgery from a distance (e.g., battlefields, space stations) [Mack, 2001]. However, the advantages afforded by robotic hardware could not be ignored as a general replacement for laparoscopy. Ideally, robots can transcend human abilities in information gathering (sensing), dexterity enhancement, and flexibility in delivery options (microscopic access). By placing a microprocessor between the surgeon's hands and the instrument tip, more intuitive motions can be performed and at a finer-level of detail. Natural hand tremors can be removed and more detailed and informative images can be collected.

The introduction of tele-operated master-slave robots such as the da Vinci® surgical system from Intuitive Surgical, Inc. [Int,] has alleviated many of these issues by offering dexterous manipulation through more than 40 different types of articulated tools, binocular vision, and a separated surgeon console with good ergonomics.

1.2 da Vinci®

The da Vinci® surgical robotic system, introduced in 1999, placed a wrist on the distal end of the MIS instruments. This minor change had a dramatic effect on surgical abilities. In this way, suture actions can follow needle trajectories more naturally, a liberty laparoscopy took away, and it allowed for more general natural motions between the hand and the instrument. The surgeon sits at a console separated from the patient and controls the robot with controllers. The robot powers the instruments inside the body and uses 3D vision by inserting two high-definition endoscopic cameras which mimic human stereoscopy, delivering depth perception at a dual-display console where the surgeon's head fits while seated, for comfort during long procedures. The robot contains three arms for surgical tools, called *patient-side manipulators*, and an additional arm for the stereo endoscopic camera. As such, the robot mimics a laparoscopic setup, but is completely robotic in control.

There are more than 1800 da Vinci® surgical systems working in operating rooms all over the world that performed about 360,000 surgical procedures in 2011. High definition stereo vision helps the surgeon see the anatomy and interact with the surgical tools with great clarity. However, robotic surgery has not yet completely taken over the MIS field. One reason is that costs are too high, as the current da Vinci® SI system costs upwards of $2 million. Another (more important) reason is that to get to new locations on the patient, the surgical team needs to re-situate the robot, which is a time consuming and cumbersome procedure, prolonging already long and arduous operations. In order to avoid this, new paradigms of surgery were developed called Single Port Access (SPA) [Romanelli and Earle, 2009] and Natural Orifice Translumenal Endoscopic Surgery (NOTES) [Lehman *et al.*, 2008]. SPA and NOTES bring in the cameras and the tools through the same, single incision (or natural orifice).

1.3 Insertable Robotic Effector Platform (IREP)

The fundamental idea of SPA and NOTES is to have all of the laparoscopic working ports entering the surgical environment through the same incision. The major drawback to such a surgical approach is that the concept of "triangulation" to which laparoscopic surgeons have grown accustomed in terms of the instrumentation and skills is challenging without being able to approach the operational site from opposing directions. As such, new robotic designs are required, such as the Insertable Robotic Effector Platform (IREP) [Xu *et al.*, 2009; Simaan *et al.*, 2012], developed jointly between Vanderbilt University and Columbia Uni-

versity.

The IREP is a continuum (”snake”) robot. Continuum robots have seen increased interest by the research community because they are dexterous, innately compliant, and down-scalable as surgical effectors for MIS, SPA, and NOTES. These robots differ from traditional industrial articulated robots in that trajectories are generated by deformation of internal structures of the robotic mechanism as opposed to the relative motion of individual rigid links.

The IREP merges our enabling technologies of endoscopic imaging and distal dexterity enhancement. This robot is a completely novel design which integrates stereo vision and dexterous end effectors for surgical interventions. It consists of two 5-degree-of-freedom (DOF) snake-like continuum robot arms, two 2-DOF parallelogram mechanisms, and one 3-DOF stereo vision module. The system is capable of decreasing the complexities of SPA procedures by meeting the following demands required by robot-assisted SPA surgery: i) the robot is able to ”fold” so that it can pass through a single small incision, ii) the robot is self-deployable into a working configuration, iii) target organs and their related tissues can be manipulated with enough precision and force, and iv) the robot has a stereo vision unit for depth perception and tool tracking [Xu *et al.*, 2009].

1.4 Computer Vision Enhancements

In their current form, the previously-described technological advancements are impressive on their own and certainly enhance the abilities of the human surgeon. However, there are a wide range of enhancements where computer software and automated algorithms can further improve the surgical experience. As the robotic hardware becomes more flexible and dexterous, they also become more complicated. In order to make the transition into operating rooms, software enhancements are needed to ease the use of the robotic tools to achieve the goal of simplifying procedures rather than making them more difficult. Additionally, without the use of intelligent algorithms, this hardware is only useful as a tele-operated extension of the surgeon, and in order to begin adding any type of automation into the procedure we must consider artificially-intelligent software add-ons, such as those afforded by sophisticated computer vision algorithms.

Surgical tool tracking is one such example which provides augmented situational awareness. Knowing the locations or the full poses of tools in the endoscopic video can enable a wide spectrum of applications. Virtual measurement capabilities can be built upon it which provides accurate measurement of sizes of

various anatomical structures. Virtual overlays indicative of the status of the tool (e.g., the firing status of an electro-cautery tool) can be placed on the instrument shaft at the tip of the instrument which is close to the surgeon's visual center of attention, enhancing the safety of using such tools. The knowledge of the tool locations is also useful in managing the tools that are off the screen, increasing patient's safety, or can be used for visual servoing to expand the effective visual field of view, in the case of motorized camera systems.

In the case of the IREP, we seek completely automated behaviors for routine actions, such as suturing, blood-suctioning, cutting, needle biopsies, or inserting/removing items to/from the body. Because of the complexity of the control of these flexible robots, commanding an accurate position to the IREP is only possible with real-time feedback about its configuration as it moves. This is because continuum robots suffer from lack of accuracy due to friction, extension and torsion of their actuation lines, shape discrepancy from nominal kinematics, and actuation coupling between segments. The safest and most cost-effective way to achieve such feedback is using vision, as hardware alternatives such as magnetic trackers are expensive, complicated, and must be sterilizable. Estimation of the shape of the continuum arms as well as the pose of the end effectors can be fed back to the robot control system to provide high accuracy and careful manipulation in an automated fashion. This may free up the surgeon's attention to other more complex tasks as the robot automatically completes these routine tasks.

1.5 Thesis Contributions

The main contributions of this thesis relate to the development and implementation of new computer vision tools to assist the surgeon during minimally-invasive laparoscopic procedures, in both the manual and robotic variants. The thesis is split into two main parts: **Tool Tracking** and **Continuum Shape Estimation**. For the tool tracking contributions, I consider three realistic, but separate, scenarios where different types of algorithms may be needed. I don't believe that there's one main approach which will solve tool tracking in all possible situations. Rather, depending on the system being used and the requirements, I consider different pros and cons that an algorithm must contain to succeed in the given use-case.

In the first case, I describe an algorithm called *Feature Flow* (Ch. 3), which is a 2D image patch tracking algorithm that learns the appearance of the surgical tool on-line, for either manual or robotic procedures. It requires only a simple initialization up-front, specifying the *(x,y)* location as well as the width and height of the bounding box specifying the tool tip in the first frame of the video sequence. From that point forward,

the algorithm learns and adjusts on-the-fly to keep track of the tool in a typically dynamic and difficult surgical environment. In this way, 2D tracking is achieved with minimal up-front information and so the introduction of new types of tools is simple because no off-line training need occur. The 2D image location of the tool can be useful to visually servo a motorized endoscopic camera system and automatically follow the movements of the surgeon.

The second and third tracking algorithms consider a robotic system with articulated surgical tools (e.g., like the da Vinci® robot). In *Virtual Template Tracking* (Ch. 4), we again learn the appearance of the tool on-line, however the goal is to recover the fully articulated 3D pose of the tool using the kinematics and the visual information from the video sequence. This is useful when we want minimal up-front information, and using only the robot's kinematic estimates and a graphical renderer we can *fix* the kinematics, which may contain errors, and recover the full joint parameters of the tool. Next, *Landmark Classification and Tracking* (Ch: 5) presents a more rigorous off-line training procedure which learns the appearance of individual, known landmarks on the tool-tip with a classification approach and builds up the articulated pose using low-level features. This assumes more up-front information in the form of ground-truth for training purposes, and it is able to persist tracking through difficult environments.

In the case of estimating the shape of continuum robots for SPA/NOTES surgical procedures, we use computer vision algorithms to learn the space of appearance changes of a robotic snake arm as it twists and bends into different configurations (Ch. 6: single-segment, and Ch. 7: multi-segment). This off-line training procedure requires a careful collection of ground-truth information, however once trained, the algorithm can very accurately estimate the configuration of the snake arms using only a stereo camera system. The algorithm is designed to consider realistic scenarios, such as occlusions and lighting changes. This information would be used to provide live feedback to the control system of the robot as it performs automated tasks, such as suturing or blood suctioning. This opens the surgeon's attention to other, more complicated parts of the procedure.

1.5.1 Importance To The Field

I believe this work is important to the field of surgery for two main reasons. First, just as a human surgeon cannot be effective without the ability to perceptually understand the scene, the introduction of robotics to surgery will be completely limited to tele-operation without computer vision algorithms to computationally interpret the scene. Every step involved in planning an automated task will involve a visual inspection of the

environment, measuring the individual parameters, and making decisions to plan the necessary movements. The robot will not be able to do this blindly, and so this work is a step towards progressing the surgical computer vision field towards better scene understanding. For example, although a high level description of suturing a wound seems straightforward, the individual decisions necessary to implement a robotic suture action (e.g., measuring the size and location of the wound, understanding the tissue surface properties for safe needle entry, mapping out the surrounding area for safe manipulation, etc) will likely be impossible without accurate vision algorithms to measure the scene.

Second, putting aside automated tasks for the future, adding computer vision capabilities to the current state-of-the-art can only enhance the innate capabilities of the human surgeon. For example, stereoscopic endoscopes deliver a perceptual 3D environment so that the surgeon can perceive differences in depth, however no computational 3D information is available. By giving the surgeon the ability to accurately measure the distances between various points in the scene, we are significantly increasing the amount of useful information available intra-operatively. In this way, the surgeon can utilize pre-operative scans into the current surgical plan by relating what he/she sees live with previous information collected before the operation began (e.g., the precise location of a tumor which isn't visually obvious to the naked eye). Tracking tools also simplifies the camera control during a manual laparoscopy. Typically, the surgeon manipulates two surgical tools while a second person controls the camera and tries to anticipate the expected motions of the camera for the surgeon. However, even the most trained camera operator cannot always anticipate where the surgeon wants to look, and when time is a factor this can be a non-trivial issue. By automating the camera motion with an automated tracker, the surgeon can in-directly control the camera motion and cause less confusion commonly caused by human error.

1.5.2 Realism

One of the major themes I attempted to incorporate into all of my thesis work was ideas and algorithms which work in realistic scenarios. Whenever possible, I show results from real surgical data and stress-test the conditions as much as possible to reveal the strengths and weaknesses. I feel that the weaknesses of an algorithm are almost as important as the strengths, so that we have realistic expectations when employing the work described below. Surgical environments are inherently chaotic, and any successful computer vision algorithm must consider many different aspects of the environment in order to transition to a real working system.

Chapter 2

Related Work

2.1 Overview

Cameras offer a passive, inexpensive approach to measuring elements of the scene during surgery. Alternatives, such as magnetic trackers, are expensive and may be difficult to sterilize. By using computer vision techniques, we are presented with less of the challenges due to autoclaving and sterilization and we can be quite accurate with the right choice in algorithms. In this work, we only consider white light methods of visual analysis and are not addressing other sensing modalities, such as ultrasound and MRI.

There are many challenges in the implementation of a vision algorithm in surgical environments, including but not limited to: non-rigidity of the structures of the environment, lighting changes, perspective and parallax effects, frequent exit-and-re-entry of the tools, motion blur, occlusions (from blood and other instruments), and a typically narrow field-of-view for the endoscopic camera systems.

There are two main vision areas being addressed in this thesis: Tool Tracking and Continuum Robot Visual Sensing. In this chapter, I present previous work in both of these fields to show how the history of these domains has evolved over time and where we currently stand with the state-of-the-art. Tool Tracking is broken down into several categories of approaches:

- Segmentation and Markers
  - Natural Information
  - Custom Color Markers
  - Segmentation Applied to 3D

  - Custom Patterns for 3D

- Marker-less Approaches

2.2 Tool Tracking

2.2.1 Segmentation and Markers: Natural Information

When using segmentation to perform tool tracking, the main goal is to label pixels in the image as either belonging to the tool or to the background. Typically, some kind of classification approach is used to learn the appearance of the tool, either online or offline. One early example is described in [Lee *et al.*, 1994], in which the authors use natural colors from the scene to classify pixels as instrument vs. organs. The method described is to train a Bayesian classifier on a large sample of typical endoscopic sequences using color statistics for classification. Temporal constraints were imposed by propagating labels from previous frames to ensure smoother pixel labelings, and holes were filled-in by leveraging the proximity of nearby labeled pixels. Finally, the labeled segmentations were fit to a bounding box to represent the shape of the tool's tip. The results of this work were evaluated against manual ground truth data and an error of $\leq 5\%$ of the image dimensions was reported. This segmentation tracker was used to center a surgical tool with visual servoing, and was achieved to within 2% of the image size.

Segmentation can be achieved without learning, as in [Doignon *et al.*, 2004; Doignon *et al.*, 2005], where surgical instruments are assumed to be metallic, leading to purely "grey" regions. In this work, boundaries of uniformly grey regions are extracted by exploiting color saturation, which was redefined to detect achromatic pixels. Using a histogram of this modified saturation definition, a probability distribution function (PDF) is used to classify pixels as tool vs background. Region growing operations from seed points at the boundaries of these regions fill-in the rest of the tool in the image, assuming that all surgical tools enter the image frame from some edge of the image. Then, cylindrical regions are used to eliminate false positives by enforcing shape statistics. Although this was shown successful, with modern day tools and the vast changes in lighting conditions, grey assumptions are not sufficient to detect and track many types of tools.

2.2.2 Segmentation and Markers: Custom Color Markers

Segmentation can also be applied to extract customized fiducial markers which are placed on the shaft or tip of the tool to assist in tracking. One such approach is described in [Groeger *et al.*, 2008; Wei *et al.*, 1997], where a custom marker is designed with a color that is identified to be absent in typical laparoscopic imagery. By collecting training data in the Hue-Saturation-Value (HSV) color space, color histograms are used to identify regions of the color spectrum which are missing throughout surgical imagery. Then, a custom marker using this color is attached to the shaft of the tool near the tip, and HSV-thresholding along with low-pass filtering is used to detect the center of this marker. The idea is that by employing a color that is not typically present in common surgical imagery, the color marker is easily identifiable as it will stand out quite clearly. Qualitative evaluations are presented in the form of surveys of surgeons both with and without robotic assistance for servoing with these color markers, and it was reported that a longer setup time is required to use these motorized camera systems, however shorter operation times are achieved.

2.2.3 Segmentation and Markers: Segmentation Applied to 3D

Next, we consider how these segmentation approaches can be used as a front-end to model-based tracking algorithms. Here, the goal is to recover the full 3D pose of the surgical tools, where the segmented pixels are used to fit to prior geometrical models in the form of cylinders. Many of the model-based approaches require a *pre-calibration* step to identify the insertion point of the tool (e.g., where the tool enters the abdominal wall). In [Doignon *et al.*, 2006], the insertion point is recovered using the accumulation of several frames in a small temporal window. First, a region-based segmentation of the pixels is performed from the edge of the image to identify those pixels which belong to the tool. By searching for low hue values and region-growing from seed locations, a 2-class line fitting procedure identifies the boundary lines of the tool's shaft. These 2D lines are collected across several frames and intersected in 3D by fitting a cylinder to the boundary lines (with a known diameter, a cylinder can be fit in closed-form to boundary lines). Because the tool's insertion point doesn't change over time, by intersecting these lines the common insertion point can be deduced.

[Nageotte *et al.*, 2009] presents a survey of four different techniques to similarly recover the insertion point:

- **3DM**: Given the two apparent lines (edges) of the cylindrical shaft, the 3D position of the axis can be estimated in closed-form. This is represented as the position of the point of the axis and the unit

vector describing this axis. In this method, the set of all axes are intersected and that which minimizes the distance to the set of axes is chosen.

- **2Dt3D**: The projection of the axis of the instrument is estimated in each image as the median line of the edges of the cylindrical contour boundary lines. Then we minimize the distance of the insertion point projection to the median lines, which results in the cylinder's axis. Finally, **3DM** is used to obtain the 3D insertion point location.

- **2D+3D**: Similar to **3DM**, but instead the planes of view are used and represented by their normals. A plane of view contains the camera center O and the insertion point of the instrument F. This belongs to a beam of planes whos axis is (OF). To find the instrument axis, a minimization of scalar products with the normal to the planes is used. Then, with these axes in mind, **2Dt3D** is used to compute the insertion point.

- **MM**: Each of the previous three methods can be used to initialize an optimization procedure which seeks to minimize the error between the edge lines of the cylinder boundaries and the perspective projection of the instrument in the estimated pose. Here we are looking for the position of the fixed insertion point F and the orientation simultaneously (e.g., the cylinder projection). This method requires a good initial guess to converge because there are numerous local minima.

Once the insertion point can be found, knowledge of this location can be used to confine the search space and detect the instrument efficiently. The authors in [Voros *et al.*, 2007] use the known location of the insertion point and represent the cylindrical shape of the tool as a point at the tool tip and three image lines: two edges along the boundaries and a symmetry line down the middle of the shaft. First, edge pixels are recovered and then all edge points with opposing gradients are used to retrieve the bisector as the cylinder axis. These edge points are then divided into those above the mid-axis and those below. A 7-10 tip pixel error is reported with 5mm of precision for the insertion point, and the system runs at 10 frames/second using quarter-sized images. However, each time the robot moves a re-calibration step is required. In this way, common factors such as patient breathing and slippage would be an issue that must be dealt with.

Another approach, presented in [Wolf *et al.*, 2011], parameterizes the abdominal wall as a spherical grid using the known insertion point. This is done using a discretized, hexagonal geode at the insertion point, where each hexagon represents a candidate pose of the tool. Using a particle filter, the pose (represented as a pan/tilt from the insertion point) can be recovered where each cell is a 3D unit vector through the

insertion point. Particles are back-projected into the image according to the candidate pose estimate and are scored based on gradient comparisons. Finally, the tool's tip is found by sliding along the best pose axis and detecting a large jump in luminance value, indicating the point where the dark shaft meets the metal tool tip. Using approximately 2000 particles, an 8.3° mean error in 3D angle is reported, corresponding to a 2.6° error in 2D pixel angle and 28 pixels of error in the position of the tool's tip.

2.2.4 Segmentation and Markers: Custom Patterns for 3D

Rather than using the insertion point to confine the search space, a known marker can be used to deduce the pose of the tool. The geometry of the pattern on the marker is known and by fitting 2D image observations to the known 3D geometry on the marker, 3D information of the tool can be deduced. One application of this is described in [Nageotte *et al.*, 2005], where the authors describe a method which localizes an instrument holding a circular suture needle using a marker and specialized colors. The needle holder is modeled as a cylinder and a marker is placed on the shaft of the tool. A colored needle (using the method in [Wei *et al.*, 1997]) is detected and an ellipse is fit in the 2D image. Using a grasping constraint, the full pose of the marker is retrieved by thresholding the luminance component of the pixels to detect the marker pattern in the image. The system runs at 3 seconds/frame and is able to localize the instrument's trocar position to ~ 2 mm. However, the algorithm needs a good guess to initialize the pose optimization procedure.

Another example of fixing a marker for 3D extraction is presented in [Krupa *et al.*, 2003], where LEDs are placed on the instrument and projected laser spots with collimators are used to recover 3D tissue surface properties. A laser pointing surgical instrument is created with 3 LEDs at the instrument tip as well as 4 redundant laser markers which blink in sync with the video frequency. An edge detection algorithm is used to detect the LED markers, and then the markers are subsequently blurred out so that the surgeon does not get distracted by them . The 3 LEDs are collinear with respect to the center of the laser spots, and so a cross ratio using a projective invariant is used to retrieve depth information from the tool's tip. Results are reported in terms of number of seconds required to center the tool with a visual servoing system, at ~ 4 seconds. Alternative marker designs are presented in [Zhang and Payandeh, 2002] with 3 vertical stripes that traverse the known diameter of the tool, which also allows for depth recovery of the marker pattern, and in [Zhao *et al.*, 2009a; Zhao *et al.*, 2009c] where a series of dots and stripes encode a bit pattern for 3D pose recovery of the tool's shaft.

There is a common problem with the aforementioned marker approaches: no matter what type of marker

is used and what kind of image processing technique is employed to extract the marker, occlusions are almost guaranteed to occur, given the small sizes of the markers and the narrow field-of-view of the cameras. To alleviate this, marker-less approaches were considered which take advantage of more parts of the tool and increase the likelihood of detection and tracking.

2.2.5 Marker-less Approaches

Marker-less approaches are rare in the computer vision tool tracking literature, as this is a much more difficult problem to achieve in real surgical environments, and as such this is the main focus of the work presented in this thesis. In this category, there are two works of interest which I discuss. In the first example [Burschka *et al.*, 2004], template matching of a small image patch of the tip of a robotic tool on the da Vinci® surgical robot is used to match in stereo imagery using cross-correlation to estimate the approximate 3D location of the tool's tip.

In other work, learning has been used to combine multiple features together into a strong feature framework [Pezzementi *et al.*, 2009], wherein the authors extract color and texture features and train offline. Here, articulated 3D tool tracking is achieved from multi-feature learning and silhouette matching by training multiple features (e.g., Red-Green-Blue, Hue-Saturation-Value, Laplacian-of-Gaussians for texture) with a Gaussian Mixture Model. Different parts of the tool are considered as separate classes (e.g., shaft, tip, background), which yields class-conditional probabilities at every pixel in the image. Then, a kinematically-driven graphical renderer matches virtual silhouettes with those obtained from the shaft and tool tip classes extracted from this pixel labeling and the approximate kinematics are recovered which best matches these silhouettes.

2.3 Continuum Robot Configuration Estimation

Although the focus of this thesis doesn't cover the specific design paradigms and associated challenges of various continuum robots, there have been different designs and actuation modalities proposed in the literature [Hirose, 1993; Simaan *et al.*, 2004; Camarillo *et al.*, 2008b; Webster III *et al.*, 2009; Dupont *et al.*, 2010; Kesner and Howe, 2011]. However, all suffer from lack of accuracy due to friction, extension and torsion of their actuation lines, shape discrepancy from nominal kinematics, and actuation coupling between segments. Therefore, it has become a priority of the continuum robot community to address these accuracy

issues with various methods.

Researchers have tried to overcome the kinematic accuracy issues of continuum robots with off-line model-based methods [Xu and Simaan, 2006; Simaan *et al.*, 2009; Agrawal *et al.*, 2010; Kesner and Howe, 2010] or off-line vision-based approaches [Gravagne and Walker, 2002; Hannan and Walker, 2003; Jones and Walker, 2006; Walker *et al.*, 2006; Camarillo *et al.*, 2008a; Croom *et al.*, 2010]. Similar to the motivation of achieving tool tracking with computer vision, by using cameras to accurately estimate the shape or configuration of continuum robots, we can provide a safer, inexpensive approach over more complicated hardware, such as magnetic trackers. However, work has shown that given sufficient feedback, accurate control of a snake robot is achievable [Bajo *et al.*, 2011]. In this work, a tiered real-time controller that uses both extrinsic and intrinsic sensory information for improved performance of multi-segment continuum robots is presented in which the higher tier of the controller uses configuration space feedback while the lower tier uses joint space feedback and a feed-forward term obtained with actuation compensation techniques. This work requires extrinsic feedback on the segment configuration, an application of the proposed work of this thesis.

2.3.1 Vision as Feedback

In [Hannan and Walker, 2003], individual vertebrae are extracted along a snake arm using gray-scale image thresholding to obtain candidate pixels along the snake arm at control points. Regions are collected from labeled pixels using a connected-components routine and successive 2D circles are fit to the ordered control points to represent the curvature of the snake arm. Ground truth of the curvatures were determined by the change in cable length, and the results of this work produced errors on the order of 20%. Out-of-plane motion was not considered, as the curve fitting would need to be extended to ellipses, and so this algorithm would need to be modified to deal with non-planar motion of the snake arm with respect to the camera, which is a common occurrence in real situations.

More accurate results are presented in [Camarillo *et al.*, 2008a], in which a voxel-carving strategy is applied to extract the position of a flexible manipulator using 3 orthogonal cameras spread about the environment. This system uses a ”shape-from-silhouette” idea in which each of the 3 cameras are placed at different viewing angles in order to extract a binary silhouette of the manipulator against a solid background. To obtain the silhouettes, the authors use a color-matting approach by subtracting the green channel from the background image and thresholding the result. A morphological filter fills-in missing pixels and then the

foreground pixels are grouped together as a single region using connected-components labeling. The visual hull is then computed from these orthogonal silhouettes by dividing the common 3D volume into a discrete 3D grid of voxels (e.g., volumetric elements). Each voxel is back-projected into each of the cameras for real-time processing, and the mutual intersection of the manipulator is achieved. Sub-millimeter root-mean squared error is presented, however the orthogonal camera setup presents an unrealistic scenario to achieve in real surgical environments.

Learning algorithms have also been used, as in [Croom *et al.*, 2010] where Self-Organizing Maps (SOMs) are applied in a stereo vision framework to detect the shape of a continuum robot without the use of fiducials for positional accuracy. In this work, the algorithm computes two stereo binary images of a black flexible manipulator against a solid white background and labels connected pixels on the robot. By initializing a SOM with a 3D point cloud (representing the physical presence of the robot), a set of reference vectors is chosen which yields an adjacency matrix on the nearest and second nearest reference vectors for each 3D point in the point cloud. The algorithm trains the reference vectors by updating each to a new position based on the average location of nearby points, and 2D projections of the 3D reference vectors are trained in parallel from the stereo disparate views. The projections are triangulated into 3D and then back-projected into the camera views for each successive training step. Accuracy was evaluated using a polynomial curve fit to the 3D backbone points and a mean error of 2 mm was achieved. Although the snake configuration accuracy is certainly sufficient for extrinsic sensory feedback, the up-front image processing algorithm would need to be upgraded for more realistic surgical scenarios.

Part I

Tool Tracking

Chapter 3

Feature Flow

3.1 Motivations

The tool tracking work in this thesis is divided into three categories. The first category, described in this chapter, is meant for a situation where 2D patch tracking (e.g., tracking a bounding box in a single 2D image) is desired, however we require only minimal information about the scene and the tools up-front. In other words, this tracker must *learn on-the-fly*, both the appearance of the tool and the background surgical scene. We call this tracking approach **Feature Flow** (FF) [Reiter and Allen, 2010].

The conditions of tracking in-vivo during minimally invasive surgery are quite different from other domains where tracking algorithms are actively being developed, such as in the mobile robot or surveillance domains. The challenges provided by working in a tight space inside the body are numerous and difficult to overcome. Changing lighting and severe pose changes are common. Because the camera is situated very close to the workspace, small 3D movements translate to large 2D pixel displacements in the image frames, and these typically occur quite quickly. Also, accounting for camera motion is challenging due to the deformable surfaces of organs that are frequently moving. The movements of the surgeon that we wish to track are often very erratic, and due to the limited field-of-view, exiting and re-entering the scene must be accounted for, which breaks continuous tracking algorithms that don't learn and can't recover.

Our approach [Reiter and Allen, 2010] to improving the state-of-the-art in 2D tool tracking seeks to combine the ideas of using multiple features with an online approach to both discriminate the object from the scene as well as account for changes in the object's appearance over time. This particular work has successfully been licensed to a start-up medical imaging company. Unlike the work in [Pezzementi *et al.*,

2009], we adjust feature representations online which allow us to discover new parts of the object as it moves and turns in a dynamically-changing environment. The basis of our algorithm lies in the following 3 assumptions:

1. One feature alone will not suffice for long-term, robust tracking.

2. Features *working together* (**synergy**) are better than features *working alone*. In this sense, features will **flow** into one another (rather than work independently and force their combination in the final step). We call this idea **Feature Flow**.

3. Learning feature representations online is important for long-term, robust tracking.

We keep a database of track states over time, storing gradient-based features (i.e., corners, edges, etc.) of the object being tracked. We use these feature locations to estimate the position of the object in the current frame (similar to [Yin and Collins, 2007]), and then compute a likelihood map using a combination of other synergistic features (i.e., color, texture). The **likelihood map**, defined as an image specifying the probability of a pixel being part of the tracked object, assists the tracker in identifying new regions of the object. We use the feature locations as ”seed points” in a region-growing algorithm, within the likelihood map to *flow* into new parts of the object, and then store this new information in our database.

In this way, gradient-based features become the main cue to the new track location, and color and texture features are used to refine and expand the analysis. Previous work [Marr and Hildreth, 1980] suggests that this flow of processing is more akin to human visual processing, where intensity changes (i.e., edges, corners) are the raw primitives extracted at the earliest stages of vision, and are then used as building blocks to group together using other visual clues to form more useful visual representations. If we detect a region of the image that exhibits similar color properties to the object we are tracking, but no mathematical transformation can be made to warp this region to our object, then it is likely that this region is not our object. On the other hand, if a region of the image can be matched to our object-of-interest, but the color properties have changed, it is much more likely (than the previous case) that it is the same object, but under different viewing conditions. Therefore, this order of processing was chosen carefully.

In the following two sections, we will describe our method for achieving each of these ideas and layout the framework for our tracking algorithm. While we have chosen a set of features that are tailored to our task, we note that one of the strengths of our system is its flexibility to use other feature sets within the same framework.

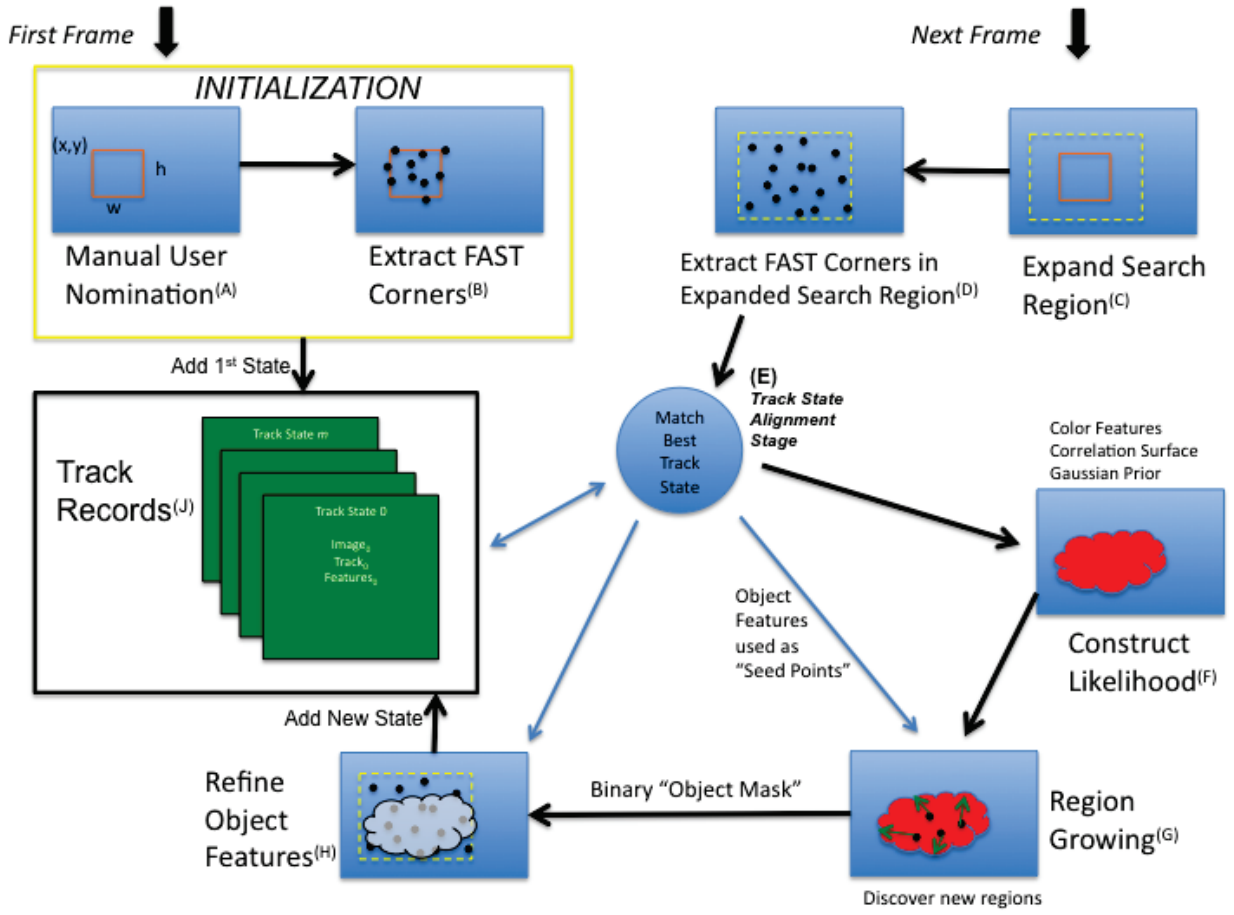


Figure 3.1: Feature-Flow tracker diagram showing the individual components of the tracker and how they feed into each other. Each module/box is labeled with a letter, which is referenced in the text to more easily visualize each step of the procedure. Small-textured features (corners) are used to estimate the track position using a database of previous track states. New regions are then discovered using features such as color and large-scale texture (correlation) in a combined, weighted-likelihood probability map. Inliers from the alignment step are used in a region-growing procedure inside the combined likelihood to identify new pixels of the object not seen before.

3.2 Overview

Figure 3.1 shows the full Feature-Flow tracker in diagram form; each module will be referenced in the text below by its corresponding letter in this diagram, for the reader's reference. We seek to keep a list of **track states** (J) to serve as a memory for the tracking system, which are separated in time, keeping a

representative set of samples of the object as it moves, and this is constantly updated as time progresses. A track state consists of: (1) the *image* frame corresponding to the time at which the track state was added to the database; (2) the *region-of-interest* representing the track location and size as it was detected in the image frame; and (3) the set of *features* representing the track state, which we use to match on each new frame. The tracker begins with a manual user nomination (A) consisting of an image patch for the target we wish to track (i.e., a surgical tool). We extract a set of FAST corners [Rosten and Drummond, 2005] within this track region *only* (B), and add an initial state to our track database. Note that if the object is not sufficiently textured, corner features are not ideal, and in this case, edges may be safely substituted.

The algorithm works as follows: on each frame, we take the previous track region and construct a slightly expanded search region-of-interest (ROI) in the current frame (C), in which we extract features (D). We want to find the best track state match (E) in the database to the current frame, and so for each track state we:

- Perform feature matching using normalized cross-correlation (NCC) of small image patches centered at each corner feature against the features stored in the track state.
- Estimate an alignment between the potential feature matches, eliminating outliers using a variant of RANSAC [Fischler and Bolles, 1981], called M-SAC [Torr and Murray, 1997].
- Evaluate the alignment by warping the track state image patch (that part of the image represented by the track stored in the state) using the estimated transform computed above, and computing the NCC as a similarity score. We use an affine model here.
- Take the track state with the highest alignment similarity score as the **best match** to the current frame.

In essence, we could stop here and track frame-to-frame using this technique. This would, and did, perform decently well if the view of the object doesn't change over time. However, as mentioned earlier, a common problem related to in-vivo environments occurs when the tool moves too much, frequently showing different views, and the current method will only be able to lock-on to that part of the object that was originally supplied in the initialization stage. The result is that we eventually lose the track because the information stored in the database is no longer present in the image frames, and the tracker has nothing else to go on to identify the object (see Fig. 3.2).

To be able to deal with this issue, we chose to combine multiple additional features together into a single likelihood map, representing the probability of each pixel being part of the object, so that we could discover

new regions of the object that are similar to what we are currently tracking. The alignment estimation step (done when we are matching to the best track state) yields a set of inlier feature locations that are presumed to be part of the object to be tracked. The inliers are used as *seed points* in the likelihood map in a region-growing algorithm, so that we can **flow** [hence the term Feature-Flow; we flow from one feature (corners) into others (those that form the likelihood)] into new parts of the object. This provides us with an object mask, which we can then use to retrieve the features associated with this image region as a new addition to the track state database.

It's important to note here that the features we retrieve from the object mask will yield new locations that weren't possible to identify with only the feature matching/alignment routines above. They are features that aren't in any of the track states in the database, but are probabilistically-likely to be a part of the object, and so we wish to keep them. This is how we learn the object online.

Optical flow was considered as the method to estimate the motion parameters of the track ROI. The most widely used formulation [Lucas and Kanade, 1981] assumes a simple translation between successive frames, which does not suffice in the in-vivo environment due to significant changes in scale, rotation, and skew over time of the tool. We don't just want frame-to-frame localization, but rather, we want to be able to match into a database with entries arbitrarily far back in time. Further modifications [Horn and Schunck, 1981] were made to minimize distortions in flow, which provides for more degrees-of-freedom. However, this is known to be very sensitive to noise. In addition, large erratic displacements tend to cause problems for even the best optic flow algorithms [Barron *et al.*, 1994], and this issue is guaranteed to be present in

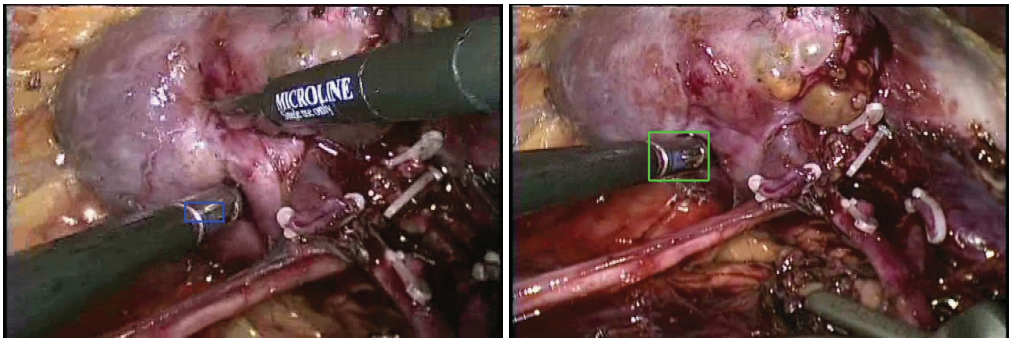


Figure 3.2: (**Left**) Without likelihood discovery, many frames after initialization, the tracker is locked-on to the same part of the object without the ability to discover new regions. As the tool turns significantly, we aren't able to pick up other regions of the same object. (**Right**) With likelihood discovery, we can capture new parts of the object.

surgical in-vivo environments. Because these techniques often require numerical differentiation to remain reasonably accurate across time, full-scale warping (as in our method) tends to provide the most accurate localization when fast motions are necessary.

3.2.1 Constructing the Likelihood Map

In the likelihood construction stage (F), we wish to define areas of the image which are likely to contain the object we are tracking (foreground) with high values and everything else (background) with low values. The input to this routine is the list of inlier features that come out of the best track state alignment along with the initial estimated track ROI.

We want the likelihood to simultaneously represent different aspects of the object to increase our chances of discriminability. Each of k features will construct its own individual, probability map, P_k, and the overall likelihood, P_i, is constructed as a weighted-sum of the individual probability maps:

$$P_i = \frac{\sum_k w_k P_k}{\sum_k w_k} \tag{3.1}$$

where w_k, for each of the k features, represents the corresponding weight on that likelihood feature map. For this work, $k = 3$; (1) a Gaussian prior representing our initial estimate of the track location from the corner feature alignment, (2) a large-scale texture analysis, and (3) a color analysis. We have initially weighted each feature equally, but note as future work research into computing dynamic optimal weights based on the filter responses. Also, other features can be used here to compute likelihood maps, with the tradeoff of computational processing; we wish to keep our tracker running in real-time, and so select the minimum number of features possible to achieve this task. It is also possible to speed up this process by utilitizing the parallel-properties of GPUs in computing these features while keeping computational efficiency [Pezzementi *et al.*, 2009].

Gaussian Prior We use a prior on the likelihood map using the current track estimate. We define a 2D elliptical Gaussian in the region where we believe the track is currently located, from the track state alignment stage (E). The Gaussian is parameterized so that it is centered at this estimated pixel location and is shaped based on the rectangular dimensions of the track match. Therefore, the mean of the Gaussian is the estimated 2D center of the track patch ROI, and the standard deviation in the x-dimension is half of the width of the estimated track patch, and in the y-dimension it is half of the height of the estimated track patch. We

also label what we believe to be the background (i.e., everywhere else) with a small, constant probability, but still a non-zero value so it is possible to recover pixels in those areas.

Correlation Surface Next, we look at large-scale texture of the object. Using the best-matched track state and the corresponding affine transform to the current frame, we create a **warped track image** (the part of the image that was stored in the track state, corresponding to the track region only, warped to the current frame) and compute a correlation surface against the current frame. Warping the track ROI is useful because typical NCC surfaces are created with the template as last seen. This could be from much earlier, and even though the small corner features were matchable, the overall structure of the object might have changed markedly. The strength here is that we have an estimate of how that track image might look like in the current frame (i.e., how it has warped), and this produces a much more accurate correlation surface. We convert this into valid probability values by setting all negative correlation scores to zero, and then we update the overall likelihood map.

Color Features Finally, we wish to exploit color features using [Collins *et al.*, 2005] to compute a set of "tuned" features from a collection of seed features on the current frame. Linear combinations of the RGB color space are used to compute local image window histograms forming the initial seed features. To create the tuned features, we normalize the histograms of each feature descriptor by the number of elements in it, thereby forming a probability distribution function (PDF) on each feature. This is done for all candidate features, in both the foreground and background separately, to achieve class-conditional PDFs. Foreground pixels are collected within the proposed object window and background pixels are collected within a slightly expanded window border around the object box, sampled locally outside the target area. The tuned feature is formed from a log-likelihood ratio of the PDFs:

$$L(i) \quad = \quad \log \frac{\max\{p(i), \delta\}}{\max\{q(i), \delta\}} \tag{3.2}$$

where p is the object probability, q is the background probability, and δ prevents taking a log of or dividing by zero. This is designed to map object pixels to positive values and background pixels to negative values. Feature discriminability is evaluated using a two-class variance ratio, where variance for the foreground class p is defined as:

$$var(L; p) = E[L^2(i)] - (E[L(i)])^2 \tag{3.3}$$

and similarly for the background class q. The variance ratio of the log likelihood function is then:

$$VR(L;p,q) = \frac{var(L;(p+q)/2)}{var(L;p)+var(L;q)} \tag{3.4}$$

and all color features are sorted based on variance ratio and the top one chosen to exploit the current frame. The variance ratio is a scalar value which identifies the discriminability of that color feature, and so sorting on this measure allows us to obtain the most discriminate feature.

We take the color features and corresponding likelihood distribution (i.e., a look-up table of color features to likelihood values) of this top color feature, and create a likelihood image of the entire frame to see how this *best* color feature can rank pixels throughout the whole image. Again, we update the overall likelihood map with the corresponding weight for this feature.

3.2.2 Exploiting the Likelihood Map

Now that we have a composite likelihood map made up of several, descriptive features, we use the inliers that came from the best track state match alignment estimation as **seed** points in a region-growing procedure (G) within the likelihood map to eliminate further outliers (if they fall in probabilistically low areas). We also can discover new areas of the object that we haven't seen before. The inliers help **flow** into new areas of the object that we can automatically discover as being part of the same object we are currently tracking. This results in a binary mask of object pixels, which can then reduce the initial set of dense, raw features extracted from the original expanded search ROI (C), and only take those that fall within the object mask, better signifying "object features" (H). These features are the ones added to the new track state. By adding a grown region of the object, we can then capture new corner features in these new regions and store in our memory. The overall track ROI can then be updated to this new region, to start again on the next frame.

3.3 Experimental Results

Our tracker was tested using in-vivo imagery from a human partial nephrectomy. We chose a challenging sequence to test the tracker's abilities against changing lighting conditions, pose and perspective changes, rapid movements into and out of the camera's field-of-view, and changing backgrounds due to movements of the organs and bleeding. The initial track was seeded manually by the user providing both a position and size of an image patch to track containing a surgical tool.

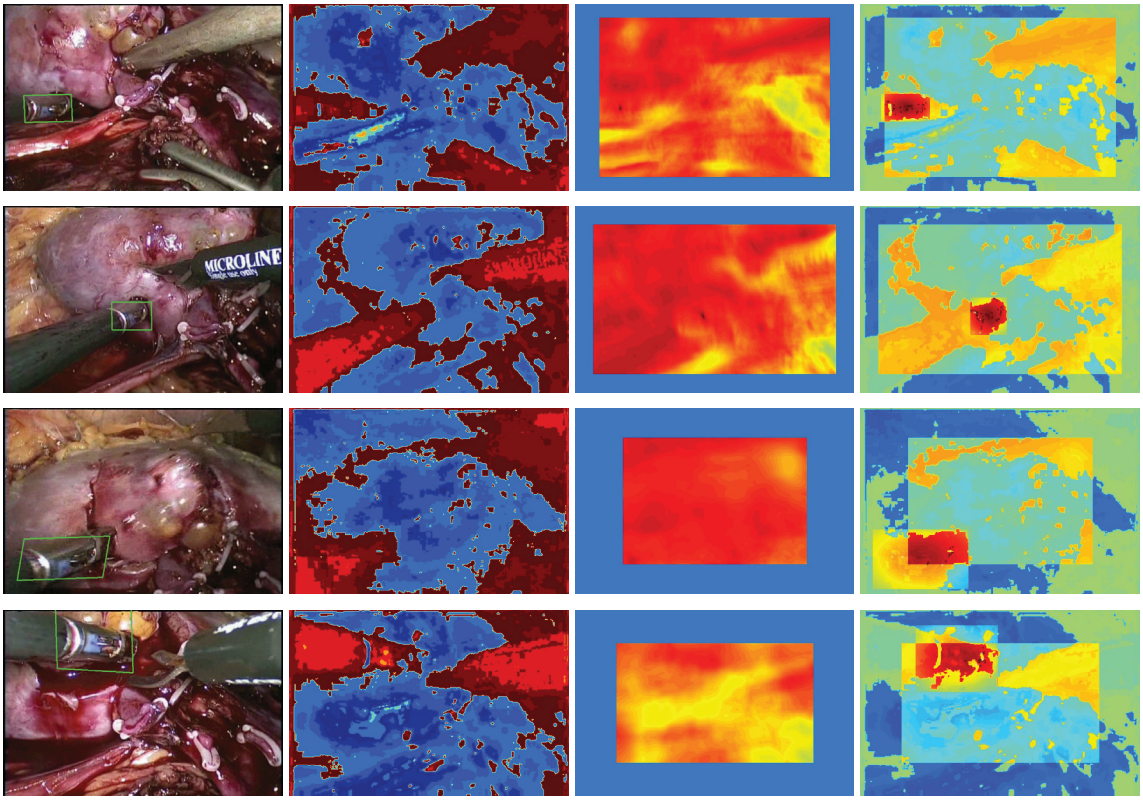


Figure 3.3: Four screenshots of tracking a surgical tool in a video sequence from a partial nephrectomy. For each of the likelihoods, red pixels indicate high probability values while blue pixels indicate low probability values. **(Column 1)** The tracked frames, with a green box drawn around the tool as final output from the tracker. **(Column 2)** The color feature likelihood images. **(Column 3)** The correlation surfaces (the blue borders are due to padding outside the window). **(Column 4)** The combined likelihoods (from weighted sums of the individual probability maps) used in the region growing procedure. We show results through severe pose changes, changes in scale, and partial occlusions. In addition, the background is changing as a bleed occurs (bottom row), but the track remains on target. The total time of tracking was 2 minutes and 17 seconds, and ended when the tool was taken out of the body, and hence could no longer been seen by the camera.

We set a maximum number of m (typically 4-5) states to serve as a ring buffer, and we define a constant time step δ ($\sim$1.5-2.5secs) to serve as the update rate for adding new states. We add new states until the memory is full, and then replace the oldest entry. This method works well on a frame-to-frame basis in a real-time setting, with efficient computer memory usage. More complex memories could be used to

increase performance, such as a log-distribution in time to keep both a long-term and short-term memory, or extracting something unique about the object (i.e., a 3-D viewpoint) and keeping a model of the object spatially, rather than temporally.

One of the strengths of this system is the ability for it to perform **track recovery**. Using the last known location of the object, we construct a slightly expanded search region ($\sim$1.5 times the previous dimensions) to look for the object. We also take into account previous consecutive failures, and grow the search region dynamically through time so that it gets larger as we miss the object more; this becomes important for tool recovery when the track is lost, is moving fast, or leaves the camera's field-of-view.

Our goal was to keep the tracker running at real-time rates, which we tested on a standard PC running Windows XP with a 2.3 GHz quad-core processor. The tracker runs at approximately 12fps, depending on the size of the object patches in the database as well as the number of states in the database at any given time. However, we found that the tracker was robust enough in finding the object of interest that we could actually skip several frames between updates; therefore, processing every third frame gave us the same qualitative results as processing every frame, effectively tripling the run-time speed to beyond standard video rate of 30Hz.

In figure 3.3 we show the individual and overall likelihoods (columns 2-4) as well as the tracked frame (column 1, on the left). The colormap for the likelihoods is designed to map high probability values to red colors and low probability values to blue colors. This sequence also required recovery several times as the tool exited the frame and then returned, and each time we were able to recover due to the dynamically-expanding search window.

Table 3.1: Performance Comparison of Different Features

Features	Continuous Tracking Time
No Likelihood Discovery	1:01
NCC (No Warp)	1:16
Color (Alone)	1:44
NCC (With Warp)	1:55
Color and NCC (With Warp)	2:17

We performed a study on this video sequence to show the effects of using a synergistic feature likelihood to discover new parts of the object on overall tracking performance, in terms of how long the system was

able to keep track. Table 3.1 shows the overall tracking time achieved with different choices of features, each starting from the beginning of the video sequence and run until the track is lost or the sequence terminates. The first row is with the corner tracker alone (i.e., no likelihood discovery stage). This is the case mentioned above, in figure 3.2, and after 1 minute and 1 second, the track was lost completely. The second row is using the NCC as the only likelihood feature, but without warping the track patch to the current frame. Recall our earlier point about the improvement of the correlation surface if the track patch is warped with the estimated track state alignment to the current frame; here the system tracked for 1 minute and 16 seconds, roughly 1/2 the total video sequence time. Next, we show using the color features alone to create the likelihood (third row), where the tracker improved to 1 minute and 44 seconds, but still 33 seconds short of tracking the entire sequence. The fourth row is with the NCC feature using a warped track patch, and we received an improvement to 1 minute and 55 seconds of tracking time, but still not enough to track the entire sequence without losing the object. Finally, the last row shows the full likelihood discovery with both color features and the NCC surface produced from the warped track patch, where we were able to succesfully track the entire sequence of 2 minutes and 17 seconds. Although the warped NCC was able to produce a reasonable result, we still needed the additional color features to track through the full sequence, thereby showing the strength of the synergy of *multiple* features.

The environment was challenging due to many erratic movements, frequent exits from the imaging field-of-view, and occlusions due to moving organs or bleeds. We successfully tracked a sequence for approximately 2 minutes and 17 seconds, to show how the tracker would work in a real enivronment. See `http://www.youtube.com/watch?v=7R6172zeNTU` for a full video of this tracking sequence. The tracking ultimately ended because the tool was taken out of the body, and hence, could no longer be seen to track. Because of the track database being updated online, no drift occurs. In addition, because of the estimation of an affine transformation against the database using image metrics that normalize the lighting conditions for feature matching, we are both able to capture geometric changes due to scale and rotational warps as well as changes due to the illumination settings of the endoscope, on a frame-to-frame basis. This significantly increases our tracking abilities over longer periods of time.

A second result, shown in figure 3.4, uses Feature-Flow to track the label on the shaft-end of the scissors to cut out part of the kidney during the partial nephrectomy sequence. The initial nomination (left-most column) is perspectively and scale-wise different from columns 2 and 3, yet we are still able to keep track. Column 4 (right-most) shows some rolling of the scissor shaft, resulting in partial occlusion of the label, but

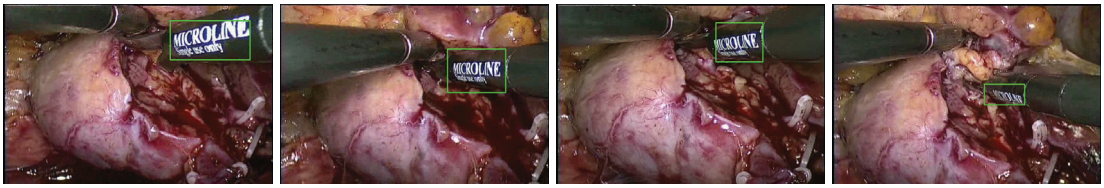


Figure 3.4: Tracking the label on the scissors during the removal of part of the kidney in the partial nephrectomy sequence. We are able to capture size and perspective changes as well as partial occlusions.

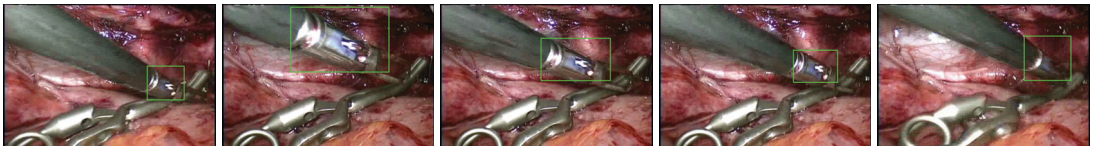


Figure 3.5: Tracking the same surgical tool as in figure 3.3, showing the tracker's ability to persist through significant size changes. The tool starts out (**Column 1**) small and slightly occluded. It then moves rapidly towards the camera and increases in size (**Column 2**), and then goes back to the same location at the original size (**Columns 3-5**), and all of these changes are accounted for.

the tracker is able to avoid being lost.

As a demonstration of the tracker's ability to adjust to size changes, we show figure 3.5, tracking the same surgical tool as show in figure 3.3, but later on in the video sequence. The tool starts off small and slightly occluded (column 1), and then rapidly moves towards the camera, growing significantly in size (column 2). It then goes back to the original location at the original size (columns 3-5), and all of these are accounted for through the full-scale feature matching and geometric warping against the track database.

3.4 Discussions

This work demonstrates a robust, long-term 2D tracking approach which addresses the real problems of an in-vivo environment. By learning both the appearance and the local environment on-the-fly, the tracker is able to deal with changes in lighting and perspective effects as well as recover when the tool exits the frame. This is an advantageous approach because no learning need occur offline, and this reduces the required up-front information. As new tools are introduced into the scene, only a bounding box initialization in the first frame is requested for tracking to begin.

A tracking approach like *Feature Flow* is applicable to motorized camera systems for visual servoing

purposes to assist the surgeon in automatically centering the image frames on the tool of interest. It is designed to provide only a 2D bounding box around the tip of the tool, as this is sufficient input to a visual servoing system. However, often we may want more detailed information, such as a 3D analysis of the location of the tool's tip or kinematic joint information for robotic articulated tools. The next chapter presents an alternative approach which is applicable to scenarios where we want to again learn on-the-fly, however the kinematic joints of an articulated surgical tool are recovered, expanding the potential applications of the tracker.

Chapter 4

Virtual Template Tracking: Template-based Articulated Tool Tracking

4.1 Motivations

When tool tracking in-vivo, we may want to recover 3D information, especially in the case of the articulated tools on the da Vinci® robot. As an alternative approach to *Feature Flow*, described in Ch. 3, we developed an algorithm to detect the articulated tool in the image *as well as* determine the articulated pose according to the robot kinematic model. In other words, the image analysis uses a model-based approach to determine the joint values of all of the active joints on the tool in a single robot arm. For this category of tracking algorithms, we wish to learn on-the-fly (e.g., no off-line ground truth needed), and we will recover the joint angles for the kinematics of the robotic tool. The algorithm is able to *fix* kinematic estimates which contain errors. We call this work **Virtual Template Tracking** (VTT) [Reiter *et al.*, 2012c].

4.2 Overview

In this work, we strive to do away with using customized markers to aid in tool tracking and we want the method to work robustly in real in-vivo environments. As such, we make use only of the natural visual information which is available on the tools and in the scene to determine a full 3D pose with articulation. We combine raw kinematic data with both on-line and off-line learning techniques. We use a virtual renderer driven by a kinematic model consistent with a tool on the robot to render the tool according to raw kinematic

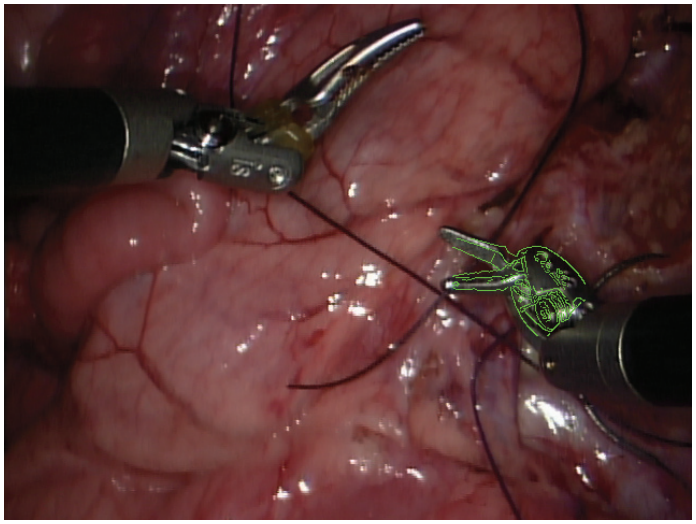


Figure 4.1: Our tool detection algorithm both automatically finds where in the image the tool exists as well as the pose and articulation according to the kinematic joint configuration. Virtual renderings which are driven by kinematic data are constructed *on-the-fly* and used to find the best matching joint configuration based on the observation from the video frame. The green overlay shows the corrected kinematic configuration according to our matching methodology.

estimates. By perturbing the tool in a small range of joint values around the kinematic estimate, we can re-render the tool at each potential joint configuration and extract templates to match against the real video imagery. The template which matches best to the image corresponds to the corrected joint kinematics.

We apply template matching as a top-down approach to the pose estimation problem. The basic idea to applying template matching to the pose estimation problem is as follows:

- Store templates corresponding to different poses
- Match all templates against the image and find the *best match location* for each template in the image
- Sort the template responses and take that which responded best to the image
- Recover the corresponding pose for that template

We construct the templates using a robot renderer which uses graphical techniques to create virtual images of the tool from a CAD model according to specific kinematic joint configurations. This is desirable because

collecting training data becomes easier than if it had to come from video ground truth and so we can collect more of it with less effort. The advantages of this type of data generation has been shown successfully in [Shotton *et al.*, 2011]. Typical template matching uses luminance images, however this is generally not photo-realistic enough to match against real imagery [Shakhnarovich *et al.*, 2003; Ning *et al.*, 2008]. Recent work [Hinterstoisser *et al.*, 2011] (called LINE) has introduced a novel image representation which stores discretized gradient orientations at each pixel as a compact bit-string. Each template then consists of high-contrast spatially-located gradient responses which represent a particular configuration of the object. A novel similarity measure which is able to maximize the cosine of a difference of gradient orientations in a small neighborhood using a lookup-table optimized to standard memory caches allows for real-time matching of many templates to a single image. Then, the template matching amounts to comparing gradient orientations at each pixel rather than luminance values. In this way, we can create templates from virtual renderings which contain information that can be successfully matched into a real image of the same object. We refer the interested reader to [Hinterstoisser *et al.*, 2011] for more details on the LINE algorithm.

An advantage of using virtual renderings to create training data is that it is easily extensible to many different tool types. Rather than requiring an arduous manual procedure for creating training data for a specific tool (off-line), training amounts to a CAD model and a renderer (on-line). The major contribution of this work is demonstrating the ability to successfully match virtual renderings robustly to real imagery, which in the past has only been done effectively with depth imagery [Shotton *et al.*, 2011]. In addition, by generating image templates through a kinematic model, we are able to tie joint configurations to appearance changes and efficiently recover the pose through a direct image match. We note that although this method is presented for use on the da Vinci® surgical system, the approach is general to any robotic arm where templates can be virtually-rendered by kinematic data and matched into an image frame.

4.2.1 Virtual Renderings

We begin with the image frame from the endoscope of the da Vinci® robot. In this work, although stereo imagery is available only one of the cameras is necessary as this is a model-based approach. Each frame has synchronized kinematic data for all parts of the robot through an application programming interface (API) [Int,]. Figure 4.2 shows a sample video frame from the left camera of the stereo endoscope as well as the virtual rendering corresponding to the kinematic estimate for that frame. Notice that although the estimate produces a close approximation to the appearance of the tool, it is inaccurate. Note that we only use the tool

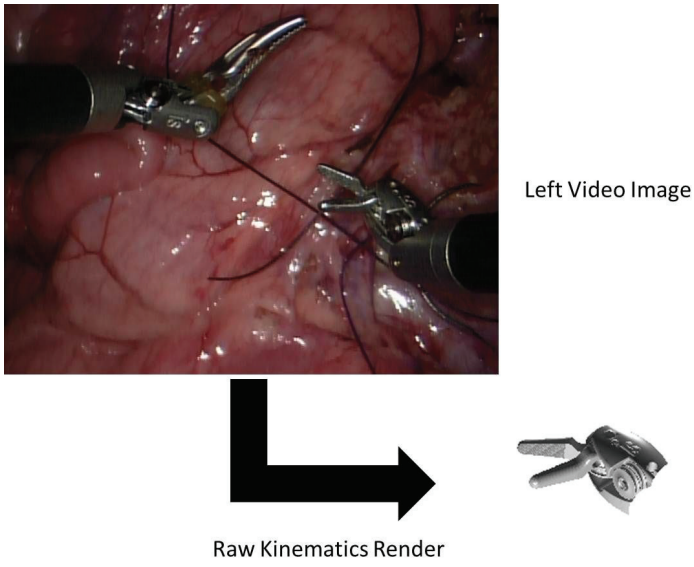


Figure 4.2: A sample image from the endoscope and the corresponding virtual render from the raw kinematics. Although the configuration is close, it's visibly-imprecise from what is observed in the video frame.

tip and hide the clevis and shaft for simplicity. In our experience, this was sufficient information to perform successful matching.

4.2.1.1 Matching

Using the virtual CAD model and the raw kinematics from a given frame, we re-render the model at nearby poses. By perturbing only a few of the active joints, we can generate renderings *on-the-fly* to match against the real image and take the kinematics estimate corresponding to the best matched template. For our experiments, we use a Large Needle Driver (LND) of the da Vinci® which has 7 total DOFs. We chose to perturb 5 of the 7 active joints, corresponding to some of the pitch and yaw parameters of the wrist and instrument as well as the roll of the instrument. However, this method allows for perturbations of whichever joints provide significant errors. We cycle each of the joints in a small range around the raw kinematics estimate and re-render the CAD model in that pose. For each, we construct a gradient template using the LINE detector and store meta-data for the kinematics with each template. Once collected, we compute a similarity measure (according to [Hinterstoisser *et al.*, 2011]) of each gradient template against the real image. For each gradient orientation in the template, a search is performed in a neighborhood of the associated gradient location for the most similar orientation in the real input image. The maximum similarity of each

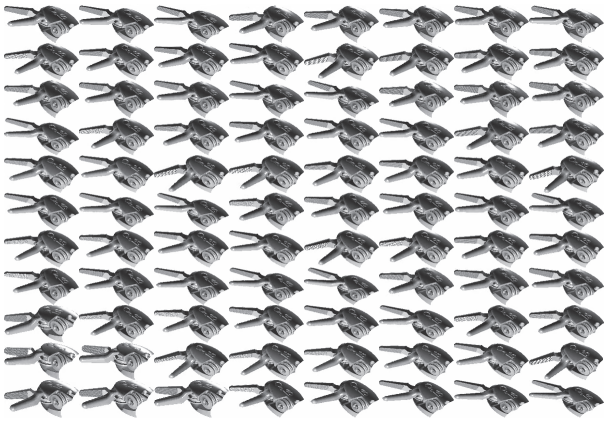


Figure 4.3: A sampling of 88 out of the total 1125 virtual renderings from the brute-force joint search matching procedure.

template to the image is stored and sorted across all templates to find the best matching gradient template(s) corresponding to the *corrected kinematics* estimate.

Brute-Force Search To perturb the joint values, we manually define a bounding box around the kinematic estimate as follows:

- Outer-yaw: $\pm 1.5°$ every $1.5°$
- Outer-pitch: $\pm 1.5°$ every $1.5°$
- Instrument-roll: $\pm 10°$ every $5°$
- Wrist-yaw: $\pm 15°$ every $7.5°$
- Wrist-pitch: $\pm 15°$ every $7.5°$

This range results in 1125 total renderings, each to be compared against the video frame. A sampling of 88 of these virtual images, displaying only the tool tips, is shown in Figure 4.3. Figure 4.1 shows the result of this search using LINE templates of each render candidate to both find where in the image the tool tip for the LND is located as well as the joint configuration corresponding to the best matching template response. The overlay is produced by only showing the edges (in green) of the best-matching template, so it's easier to observe the accuracy of the alignment qualitatively.

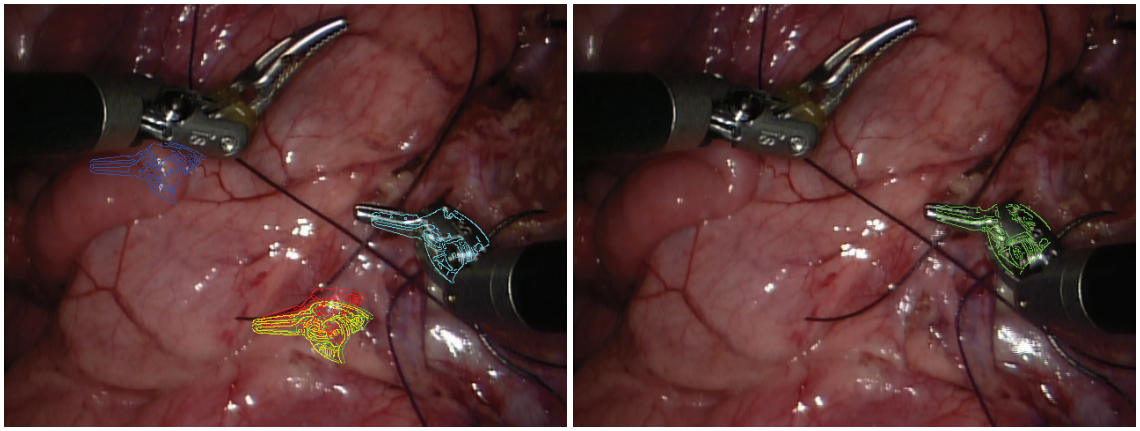


Figure 4.4: Using our likelihood rejection method, we can reject false positives [**Left**] and recover the best matched pose [**Right**] with a higher detection rate by collecting more of the top matches and eliminating those that don't probabilistically fall on the tool tip. Green labels the best match according to LINE, which is clearly off the tool in the left figure. However, after applying the false positive rejection method on the right, the top match is more visibly consistent.

It's important to note that in typical template matching approaches, scale invariance is an issue which complicates the match procedure. Other approaches tend to normalize the template and/or evaluate against an image pyramid. In our case, certain joints that we perturb physically move the tool tip closer/further from the camera center, thereby achieving candidate templates at different scales. This is advantageous because we then know the joints that correspond to those templates which is preferable to normalizing, a procedure which would lose this important joint information. In the end, we want sensitivity to different scales so that we can recover the joints that produce the proper scale of the tool in the image.

In practice, the best-matching template according to the LINE search is not necessarily always the correct answer. To account for this, we collect the top k (~ 5) best matches. It can often occur that the actual correct response is the 2^{nd} or even 3^{rd} best LINE match according to the template score. This often occurs when the top matches have the same score and are arbitrarily ordered. In these cases, what we noticed is that the better-scored matches occur elsewhere in the image, either on the other tool or on the background tissue. However, this is something that we can account for, as described in the next section.

4.2.2 False Positive Rejections

In order to increase the detection rate, we collect more of the top matches from the LINE responses. Because some of the higher responses may sometimes fall in other areas of the image (Figure 4.4, left), we developed two false positive rejection schemes.

Likelihood Rejection Using the method described in [Pezzementi *et al.*, 2009] (and code generously provided by the authors) we train off-line from manually-labeled sample video images to construct class-conditional probabilities using several color and texture features at the pixel level. We train using 3 classes to label each pixel as either on the tool shaft (class 1), metal (class 2), or tissue (class 3). The data which was used to train was from a completely separate data set than what was used for testing, to further confirm the usefulness of the method. Pixels that are labeled as metal correspond to the tool tip or clevis and allow us to filter false positives by rejecting those LINE detections which fall in probabilistically-low regions in the metal likelihood labeling. This threshold is determined empirically, however future work plans on investigating more automated methods of determining this threshold. One such possibility is to analyze a histogram of probability values and comparing to a prior based on the expected number of pixels on the tool tip and clevis.

An example of the output of this training procedure is shown in Figure 4.5. The upper-left image shows the original image frame from an in-vivo sequence containing two LND tools performing a suture. The upper-right is the *metal* class likelihood labeling, where brighter pixels indicate higher probabilities. The lower-left image similarly shows the *shaft* class likelihood labeling, and finally the lower-right shows the *background* class. As you can see, the metal labeling sufficiently marks pixels in the area where the tool detections should occur, and helps reduce false positives in other areas. The shaft labeling can also be put to use, as described in the next section.

Vanishing Point Rejection Sometimes a false positive can fall on the other tool in the image on the metal tip, where the previous rejection method would fail. To further reduce false positives, we noticed the following: the parallel lines that make up the edges of the tool's shaft boundary share a vanishing point (VP) in common. Furthermore, the shaft of the tool will always come into the image frame from the edge of the image. We can compute the VP according to the shaft orientation from the kinematics and the camera model [Hartley and Zisserman, 2003]. For each of the LINE hypotheses (each of which has a candidate kinematic joint configuration), we can identify the corresponding end of the proximal clevis (the metal part which connects the black shaft with the metal tool tip, at the wrist) as an arc consistent with that hypothesis. Then we

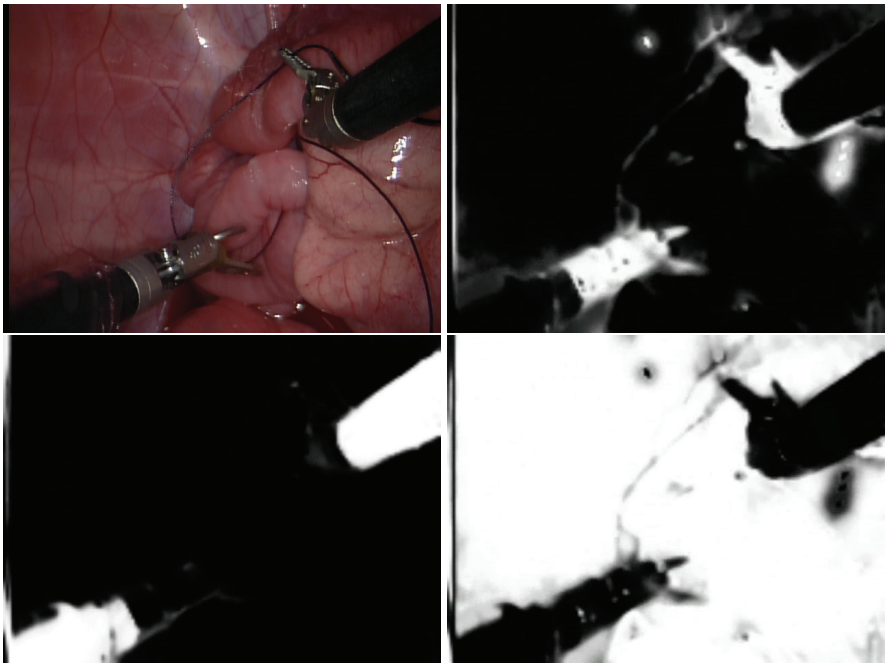


Figure 4.5: Example likelihood images from class-conditional pixel labeling after the training procedure. [**Upper-Left**] The original image from an in-vivo sequence of 2 LND tools performing a suturing procedure. [**Upper-Right**] Metal likelihood (*e.g.*, tool tip, clevis) [**Lower-Left**] Tool shaft likelihood [**Lower-Right**] Background class likelihood

connect the VP and the two ends of the proximal clevis arc and extend the lines to the image boundary. Figure 4.6 describes this idea visually. We can obtain the regions A (red) and B (green) from the proximal clevis arc and compute the overlap of those regions with the shaft likelihood labeling. Then, region B is easily rejected as a false positive while region A persists as a possible candidate match.

4.3 Experiments

Experiments were run on porcine in-vivo data from the da Vinci® surgical system. In the following sections, we discuss accuracy and timing results from our experiments.

4.3.1 Accuracy

The main failure case that we encounter is shown in Figure 4.7, where the detection occurs on the correct tool tip, but with a visibly wrong configuration. This is determined by eye as a simple counting of overlaying pixels tends to call these correct when in fact they are wrong. All other false positives (e.g., detections that occur on tissue or the other tool) are successfully rejected by our false positive rejection method. In an experiment with real in-vivo data, our brute-force method yielded an 87% correctness rate (on the correct tool tip, in a visibly-consistent joint configuration according to the template overlay). However, we obtained 11% with no detection, meaning all of the top 5 matches were rejected by the false positive method on those frames. This is a preferable result as we then have more confidence in our actual detections and would prefer a *no-response* over a *wrong response*. Finally, 2% were labeled incorrectly, with wrong configurations

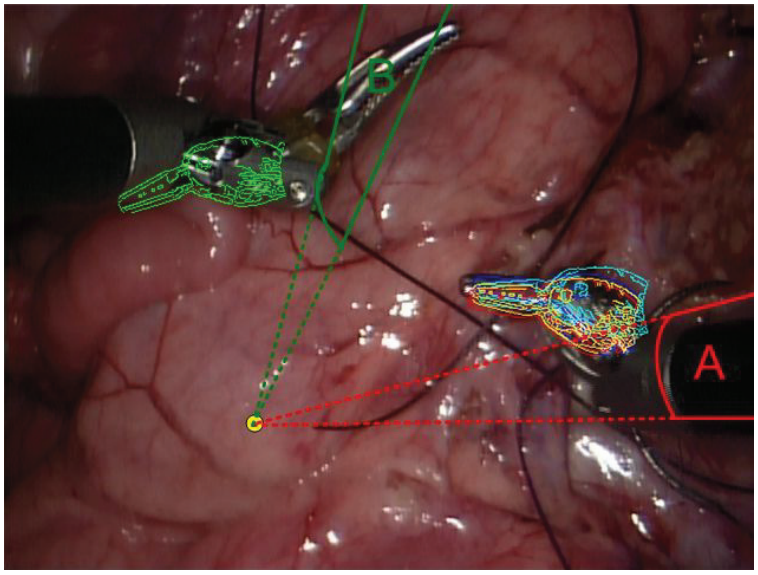


Figure 4.6: We compute the vanishing point of the tool shaft we are detecting using kinematics data and a camera model. Then we connect this with the candidate proximal clevis of each of the hypotheses according to the top LINE matches and use the shaft likelihood labeling to eliminate projectively-inconsistent kinematic hypotheses. The region labeled A in red marks the proximal clevis associated with the red LINE detection on the proper tool tip. The region labeled B in green is a false positive on the wrong tool. By intersecting these regions with the shaft likelihood labeling, region A intersects many pixels with the shaft likelihood while region B has a null intersection.

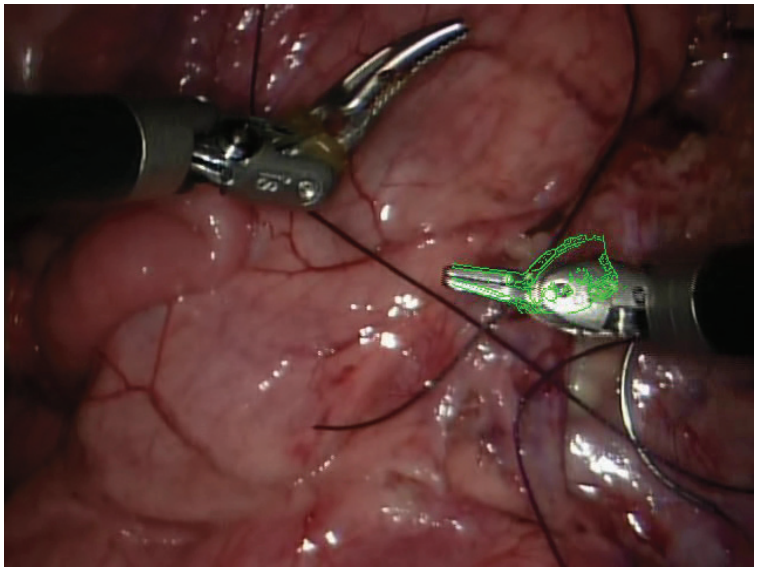


Figure 4.7: A failure case is defined as a detection that occurs on the correct tool tip with a visibly-wrong joint configuration.

(determined by eye).

From our analysis, the remaining incorrect detections tend to occur when the tool is in an ambiguous configuration, such as extended straight out or if little texture is shown. The cases that work very well are when the clevis bends at an angle from the shaft, creating a unique shape that's more easily detected by the LINE templates.

4.3.2 Timing

The brute force method renders 1125 templates per-frame, which is not a real-time solution. There are several potential speed-ups which can retain the accuracy described in Section 4.3.1 while speeding-up the processing per-frame. The tests of each template against the image frame are independent from each other, and as such are parallelizable, either on the GPU or on a multi-core/processor CPU, neither of which we implemented for this work. Other than that, the two significant bottlenecks are the *number of renderings* as well as the *time per-rendering*. We use OpenGL to render the model in particular configurations, and downloading the image from the GPU into software to process each template is computationally-expensive. Rather than downloading the entire image, we use the kinematic data to determine a region-of-interest in the

virtual image where the tool-tip will be drawn and only download that part of the graphics buffer. However, the number of renderings needs to be reduced as well. The brute force search takes ~ 23 seconds/frame to match all templates and produce the best pose. In the next section we describe an approach which yields an **87% speed-up**.

Pyramidal Search We make the assumption that the manifold of appearances for the tool tip is relatively smooth, and so a very coarse search should yield a match which is close to the actual result. We can then use this to initiate a finer-scaled search in a much smaller range and reduce the overall number of templates to render. We choose a 2-level pyramid search scheme where the actual joint search ranges were determined by hand in both a *coarse* and *fine* manner. The coarse search uses the following search range, relative to the raw kinematics estimate as in the brute-force method:

- Outer-yaw: $\pm 1^\circ$ every 2°
- Outer-pitch: $\pm 1^\circ$ every 2°
- Instrument-roll: $\pm 10^\circ$ every 10°
- Wrist-yaw: $\pm 15^\circ$ every 15°
- Wrist-pitch: $\pm 15^\circ$ every 15°

This yields a total of 108 renders in the first-level. We take the top k-matches using LINE and use the same false-positive rejection method as described in Section 4.2.2. The template with the best response to survive the false-positive rejection initiates a finer-scaled search with the following search range, relative to the joint configuration which produced the coarse match:

- Outer-yaw: $\pm 0.25^\circ$ every 0.5°
- Outer-pitch: $\pm 0.25^\circ$ every 0.5°
- Instrument-roll: $\pm 5^\circ$ every 5°
- Wrist-yaw: $\pm 5^\circ$ every 5°
- Wrist-pitch: $\pm 5^\circ$ every 5°

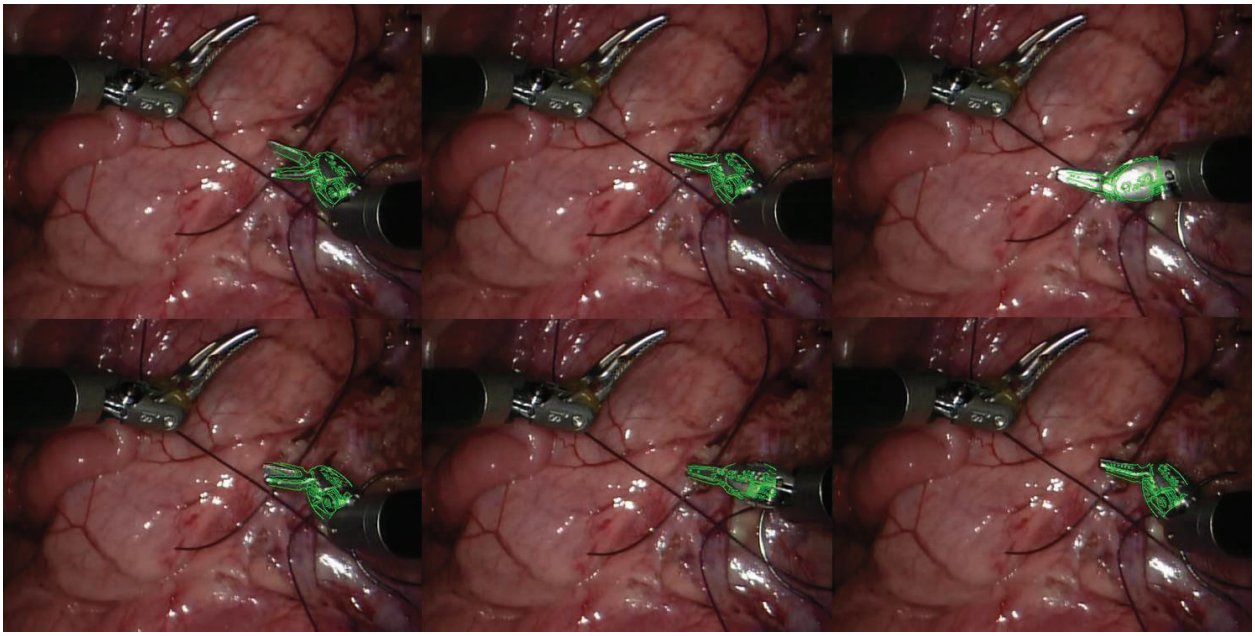


Figure 4.8: Sample output frames from running the tool detector on an in-vivo video sequence. Several different poses are shown as well as an example with a high specularity (top-right). The green overlay is produced by drawing only the edges of the best-matching virtually-rendered template, to assist in seeing the quality of the result. These results are produced from the pyramid-search method.

This produces an additional 108 renders for a total of 216 total renderings in the pyramid search, a reduction of 81% in the overall number of renderings per-frame from the brute-force method. Using the same LINE matching and false-positive rejection, we take the joint values corresponding to the best match in the fine-scaled search as the final result. This reduces the total per-frame processing time from 23 secs/frame down to 3 secs/frame, an 87% speed-up in processing time. Figure 4.8 shows sample frames from running the detector using the pyramid-search method. The green overlays show the edges of the best virtually-rendered templates (again only displaying the edges of the templates which help visualize the quality of the match). We show several different poses and articulations of the tool, as well as an example with significant specularities (upper-right image). The detection rate of the pyramid-search method was nearly identical to the brute-force method, at 86% correctness.

4.4 Discussions

The power of this approach lies in the ability to learn from virtual renderings which are kinematically-generated and accurately applying this to real imagery. This scheme has been shown to be successful in the object recognition [Ohbuchi *et al.*, 2008] and human pose recognition [Shotton *et al.*, 2011] domains where **depth images** are used to learn appearances. In the case of [Shotton *et al.*, 2011], synthetic depth images render articulated human poses to create a richer database which is not feasible to do with the amount of real data needed to accomplish the same task. In our work, we provide a means to accomplish a similar goal for matching against **real imagery** by use of a template which is created from a virtual image and is matchable against a real image.

The consequences of this are two-fold: first, we are able to learn *on-the-fly* without knowledge of the poses that can occur during the surgery, and thus can reduce the memory footprint of a database-like approach which must account for all possible poses and articulations to succeed robustly. Second, the method becomes easily extensible to different tool types. As new tools are developed for a robot such as the da Vinci®, we can easily apply our algorithm to it in real imagery with nothing but a CAD model and raw kinematics estimates.

Being a top-down method for a high-dimensional problem, computation is usually a bottleneck. There are multiple ways to further reduce the computations to bring it down to real-time. Since rendering is already done on the GPU, the run-time can be pushed down further if the generation and matching of the LINE templates are also done on the GPU, both taking advantage of parallelization and avoiding the large memory transfer from the GPU. Gradient-based search routines, such as Levenberg-Marquardt, could potentially reduce the matching time although it may not find the optimal solution. The accuracy of any optimization routine must be considered as well. We also note that the likelihood maps may have a further use to reduce the number of renderings. This pixel labeling can cue a data-driven joint search, where the search range can be potentially determined using inverse kinematics of only those configurations that are consistent with the pixel labeling of the metal-class (e.g., tool tip and clevis). This is an area for future exploration.

Although we investigate the detection problem in this work, the final system will run in a tracking mode. We notice that between consecutive frames in a video sequence there is only very little movement that tends to occur. This means that the set of virtual renderings will only need to be updated slightly, and most of the templates can be reused from frame-to-frame. This should also help to increase the detection accuracy rate by confining searches to previous successfully detected locations.

Chapter 5

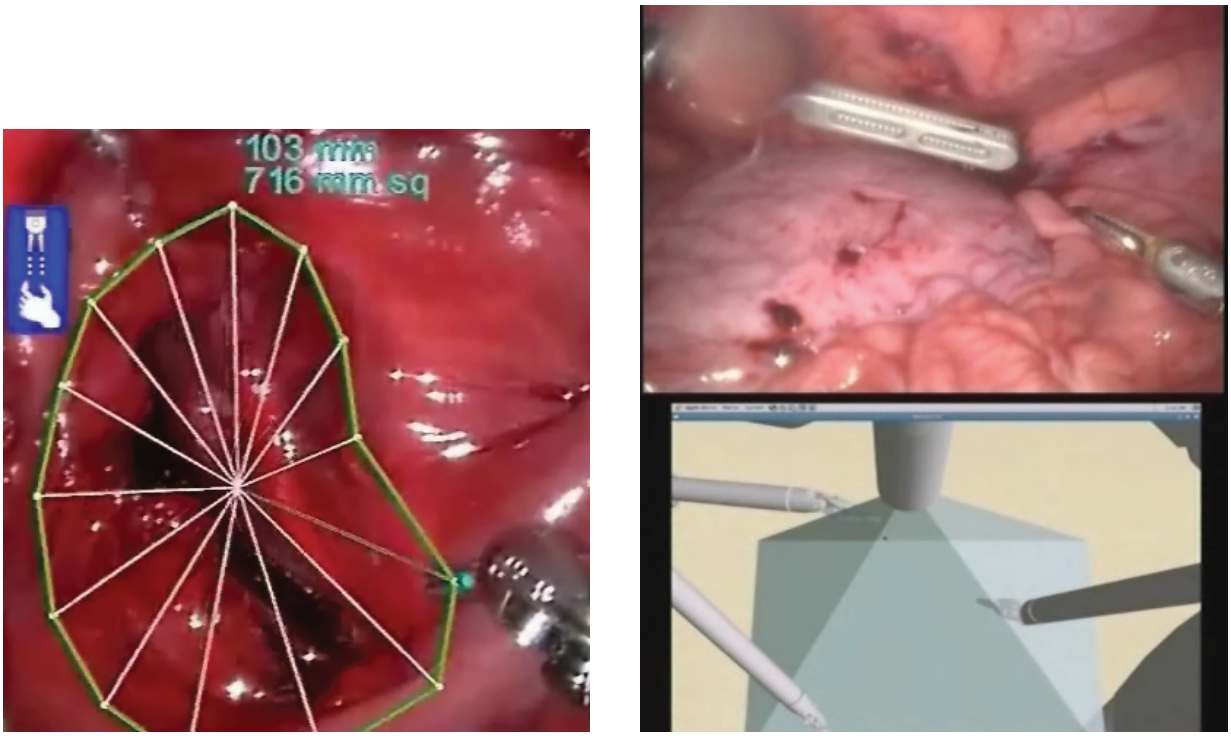
Landmark Classification and Tracking: Learning Features for 3D Tool Tracking

5.1 Motivations

The final category of tool tracking algorithms we wish to address is also applied towards 3D articulated robotic arms, however in this situation we want to rigorously learn, from off-line training data, the appearance of individual landmark features and build-up the pose in 3D from the *bottom-up*. In contrast to the VTT approach described in Ch. 4, here the algorithm works *with* the kinematics to fix errors by detecting known locations on the tools and recovering the 3D pose by applying geometric constraints and a probabilistic framework with stereo vision. We call this work **Landmark Classification and Tracking** (LCT) [Reiter *et al.*, 2012a; Reiter *et al.*, 2012b].

5.1.1 Applications

Advancements in minimally-invasive surgery have come about through technological breakthroughs in endoscopic technology, smarter instruments, and enhanced video capabilities [Mack, 2001]. These achievements have had a common goal of continuing to reduce the invasiveness of surgical procedures. Robotic hardware and intelligent algorithms open the doors to more complex procedures by enhancing the dexterity of the surgeon's movements as well as increasing safety through mechanisms like motion scaling and stereo imaging.



(a) Ruler (b) Lost Tool

Figure 5.1: Two applications of tool tracking: In **(a)**, a picture of the measurement tool measuring the circumference and area of a mitral valve is shown. In **(b)**, an example scenario of a lost tool (e.g., outside the camera's field-of-view) is shown, whereby the endoscopic image (top) shows only two tools, and with fixed kinematics and a graphical display (bottom), we can accurately show the surgeon where the third tool (out of the left-bottom corner) is located and posed so they can safely manipulate the tool back into the field-of-view.

Intuitive Surgical's da Vinci® robot [Int,] is the most prevalent example of such a technology, as there are more than 1800 da Vinci® surgical systems in operating rooms worldwide which performed about 360,000 procedures in 2011. In this system, high definition stereo vision delivers a perceptual 3D image to the surgeon which helps to see the anatomy and interact with the surgical tools with great clarity. Augmenting the surgeon's vision with other relevant information in the form of graphical overlays can further help the surgeons/patients in a different dimension. Tool tracking is a manifestation of intelligent computation which can improve the situational awareness for a surgeon during a procedure.

Knowledge of the locations of tools in the endoscopic image can enable a wide spectrum of applica-

tions. Accurate tool localizations can be used as a *Virtual Ruler* (see Fig. 5.1(a)), capable of measuring the distances between various points in the scene, such as the sizes of anatomical structures. Graphical overlays can indicate the status of a particular tool, for example, in the case of the firing status of an electro-cautery tool. These indicators can be placed at the tip of the tool in the visualizer which is close to the surgeon's visual center of attention, enhancing the overall safety of using such tools. It can also be useful in managing the tools that are off the screen (see Fig: 5.1(b)), increasing patient's safety, or for visual servoing of motorized cameras.

The joints of a robotic surgical system are typically equipped with encoders so that the pose of the end effectors can be computed using forward kinematics. In the da Vinci®, the kinematic chain between the camera and the tool tip involves 18 joints and more than 2 meters in cumulative length, which is challenging to the accuracy of absolute position sensing and would require arduous and time-consuming procedures to accurately calibrate. However, a master-slave robotic system does not require high absolute accuracy because humans are in the control loop. As a result, we have observed up to 1-inch of absolute error, which is too large for most of the applications that are mentioned above. Therefore, tracking the tools from images is a practical and non-invasive way to achieve the accuracy requirements of the applications.

In this work, we present a tracking system which learns classes of natural landmarks on articulated tools off-line by training an efficient multi-class classifier on a discriminative feature descriptor from manually ground-truthed data. We run the classifier on a new image frame to detect all extrema representing the location of each feature type, where confidence values and geometric constraints help to reject false positives. Next, we stereo match in the corresponding camera to recover 3D point locations on the tool. By knowing *a priori* the locations of these landmarks on the tool part (from the tool's CAD model), we can recover the pose of the tool by applying a fusion algorithm of kinematics and these 3D locations over time and computing the most stable solution of the configuration. Our tracker is able to deal with multiple tools simultaneously by applying a tool association algorithm and is able to detect features on different types of tools. More details on each of these steps follow in the remaining sections.

5.2 Methods

In this section we present an overview of our tool tracking method. Fig. 5.2 shows a visual overview of our detection and tracking system. Before we begin, we present an overview of the robotic hardware system as

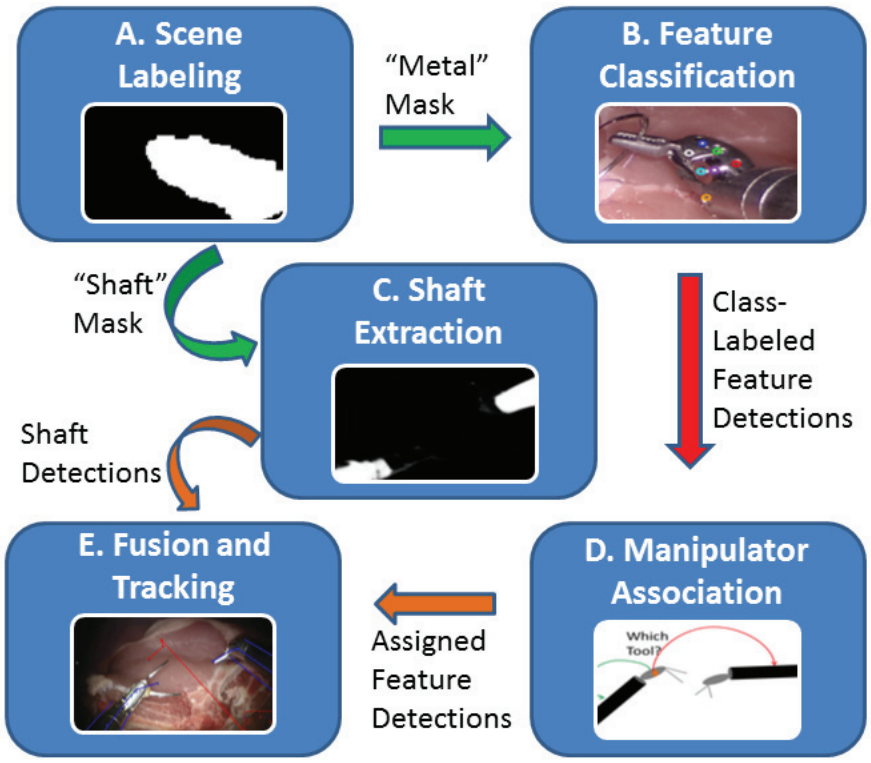


Figure 5.2: Algorithm Overview of the Tracking System: (**A**): The *Scene Labeling* module applies a multi-feature training algorithm to label all pixels in the image as one of three classes: Metal, Shaft, and Background, producing binary masks for each. (**B**): The *Feature Classification* module uses a classifier on feature descriptors to localize known landmarks on the tool tips. (**C**): The *Shaft Extraction* uses the shaft mask from module **A** to fit cylinders to the shaft pixels in the image for all visible tools, whenever possible. (**D**): The *Patient-Side Manipulator (PSM) Association* module uses class-labeled feature detections output from module **B** to determine which feature is associated with which tool in the image. (**E**): The *Fusion and Tracking* module takes outputs from both **C** and **D** to fuse visual observations with raw kinematics and track the articulated tools over time.

well as information on calibration procedures performed prior to the work presented in this chapter.

5.2.1 System Overview

The da Vinci® surgical robot is a tele-operated, master-slave robotic system. The main surgical console is separated from the patient, whereby the surgeon sits in a stereo viewing console and controls the robotic

tools with two *Master Tool Manipulators* (MTM) while viewing stereoscopic high-definition video. The patient-side hardware contains three robotic manipulator arms along with an endoscopic robotic arm for the stereo laparoscope. A typical robotic arm has 7 total degrees-of-freedom (DOFs), and articulates at the wrist. The stereo camera system is calibrated for both intrinsics and stereo extrinsics using standard camera calibration techniques [Zhang, 2000]. Although the cameras have the ability to change focus during the procedure, a discrete number of fixed focus settings are possible, and camera calibration configurations for each setting are stored and available at all times, facilitating stereo vision approaches as described later on in this chapter.

5.2.2 Scene Labeling

We begin with the method described in [Pezzementi *et al.*, 2009] to label every pixel in the image as one of three classes: *Metal*, *Shaft*, or *Background* (listed as module **A** in the algorithm overview of Fig. 5.2). A Gaussian Mixture Model (GMM) [Duda *et al.*, 2001] of several color and texture features is learned off-line for each of these three classes. Subsequently, we can assign a class-conditional probability for each of the classes to every pixel and assign a label. Fig. 5.3 shows an example result of this pixel labeling routine, with the original image from an in-vivo porcine sequence on the upper-left, the metal class on the upper-right, the shaft class on the lower-left, and the background class on the lower-right. The metal class represents all pixels located at the distal tip of the tool, from the clevis to the grippers. These are where all of the features which we wish to detect are located. Additionally, we will describe later on how the shaft class is used to fit a cylinder to the tool's shaft, whenever possible.

Typically, surgeries performed with the da Vinci® are quite zoomed in, and so the shaft is not usually visible enough to fit a cylinder (the typical approach to many tool tracking algorithms [Voros *et al.*, 2007]). However, at times the camera is zoomed out and so this scene pixel labeling routine allows the algorithm to estimate the 6-DOF pose of the shaft as additional information (Sec. 5.2.5). By estimating the approximate distance of the tool from the camera using stereo matching of sparse corner features on the tool's tip, we can estimate if the shaft is visible enough to attempt to fit a cylinder. When the camera is zoomed out, although the shaft is visible the features on the tool tip are not so easily detected. Therefore, we can pick-and-choose between shaft features, tool-tip features, and a hybrid in between depending on the distance of the tool to the camera. These pixel labelings help to assist in both feature detection and shaft detection, as described further in the following text.

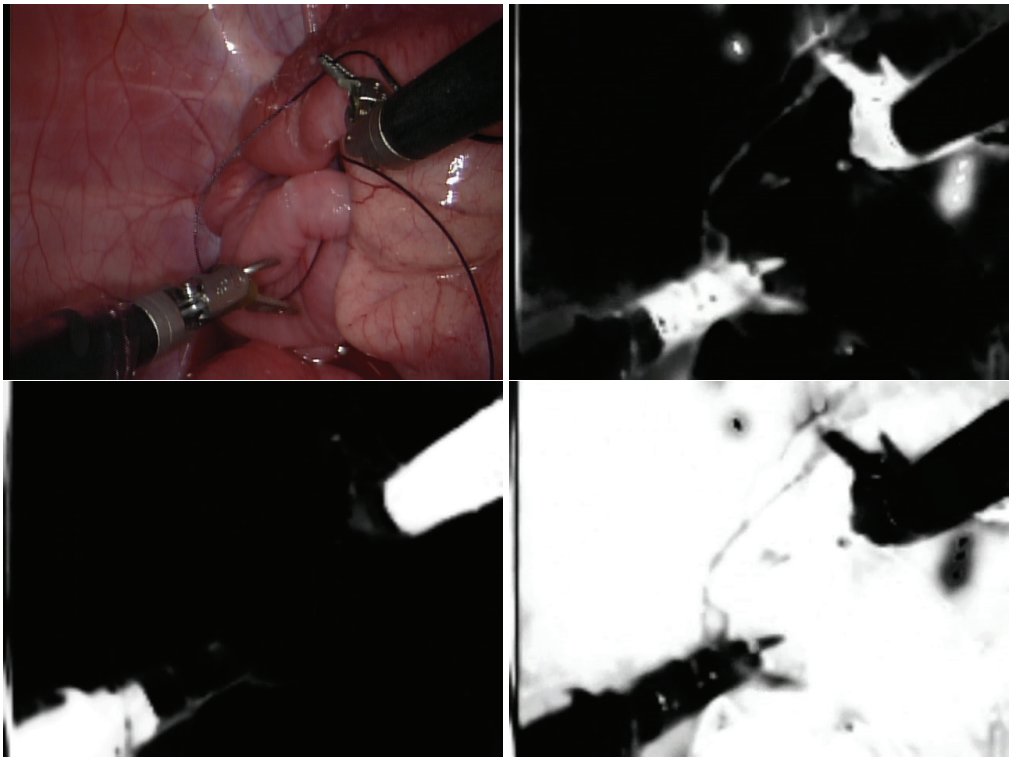


Figure 5.3: Example likelihood images from class-conditional pixel labeling as described in section 5.2.2. [**Upper-Left**] The original image from an in-vivo sequence of 2 robotic tools performing a suturing procedure. [**Upper-Right**] Metal likelihood (*e.g.*, tool tip, clevis) [**Lower-Left**] Tool shaft likelihood [**Lower-Right**] Background class likelihood

5.2.3 Feature Classification

Our feature classification (module **B** in Fig. 5.2) works by analyzing only the pixels which were labeled as *Metal*, using the method previously described in Sec: 5.2.2. This reduces both the false positive rate as well as the computation time, helping us to avoid analyzing pixels which are not likely to be one of our features of interest (because we know beforehand they are all located on the tool tip). We train a multi-class classifier using a discriminative feature descriptor and then localize class-labeled features in the image. Next, we stereo match and triangulate these candidate feature detections to localize as 3D coordinates. These feature detection candidates are analyzed further using known geometric constraints (described in Sec. 5.2.4.2) to remove outliers and then are fed into the fusion and tracking stage of the algorithm. We begin with a detailed description of each of these feature classification steps.

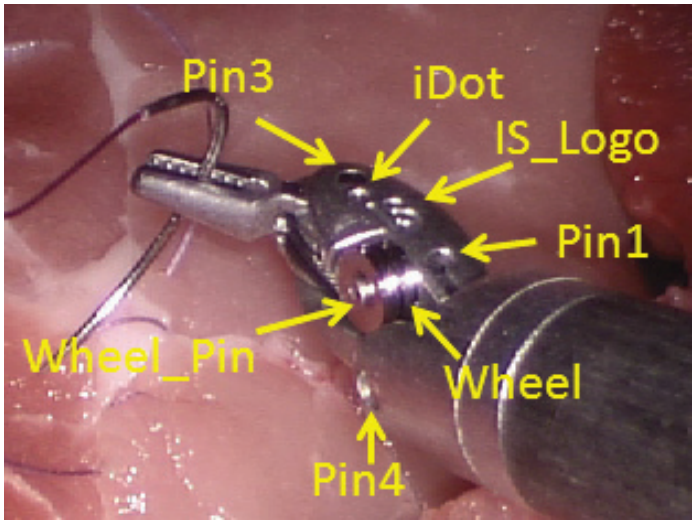


Figure 5.4: Ground truth guide for the feature classes we detect on the LND tool. We concentrate on 7 different types of naturally-occurring landmarks.

5.2.3.1 Training Data Collection

We begin by collecting data for the purposes of training our classifier. We use nine different video sequences which span various in-vivo experiments, to best cover a range of appearance and lighting scenarios. For training, we use only the *Large Needle Driver* (LND) tool, however as we will show later on this will extend well to other types of tools, such as the *Maryland Bipolar Forceps* (MBF) and *Round Tip Scissors* (RTS). Seven naturally-occurring landmarks are manually selected and shown in Fig. 5.4 overlaid on an image of the LND. The features chosen are of the pins that hold the distal clevis together, the **IS** logo in the center, and the wheel and wheel pin. For purposes of this work, from time-to-time we may refer to this combination of landmarks as a *marker pattern*, M_i. We also add known, invariant locations on the mid-line of the shaft axis (described in Sec. 5.2.5) to this marker pattern to be used in the fusion module.

For each frame in the ground truth procedure, we manually drag the best encompassing bounding-box around each feature of interest, as we want to avoid contamination from pixels which don't belong to the tool. To obtain as large a dataset as possible with reasonable effort, we coast through small temporal spaces using Lucas-Kanade optical flow (KLT) [Lucas and Kanade, 1981] to predict ground truth locations between user clicks as follows:

- The user drags a bounding-box around a feature of interest

- The software uses KLT optical flow to *track* this feature from frame-to-frame (keeping the same dimensions of the box)
- As the user inspects each frame, if either the track gets lost or the size changes, the user drags a new bounding-box and starts again until the video sequence ends

This allows for faster ground truth data collection while still manually-inspecting for accurate data. Overall, we use $\sim 20,000$ total training samples across the seven feature classes. Before we describe the classifier algorithm, we first discuss the feature descriptor which is used to best discriminate these feature landmarks from each other robustly.

5.2.3.2 Feature Descriptor

We require a discriminative and robust region descriptor to describe the feature classes because each feature is fairly small (e.g., 17-25 pixels wide, or $\sim 2\%$ of the image). We choose the Region Covariance Descriptor [Tuzel *et al.*, 2006], where the symmetric square covariance matrix of d features in a small image region serves as the feature descriptor (see Fig. 5.5). Given an image I of size $[W \times H]$, we extract d=*11* features, resulting in a $[W \times H \times d]$ feature image:

$$\mathbf{F} = [x \;\; y \;\; Hue \;\; Sat \;\; Val \;\; I_x \;\; I_y \;\; I_{xx} \;\; I_{yy} \;\; \sqrt{I_x^2 + I_y^2} \;\; \arctan(\frac{I_y}{I_x})] \tag{5.1}$$

where x, y are the pixel locations; *Hue, Sat, Val* are the hue, saturation, and luminance values from the HSV color transformation at pixel location (x, y); I_x, I_y are the first-order spatial derivatives; I_{xx}, I_{yy} are the second-order spatial derivatives; and the latter two features are the gradient magnitude and orientation, respectively. The first two pixel location features are useful because their correlation with the other features are present in the off-diagonal entries in the covariance matrix [Tuzel *et al.*, 2006]. The $[d \times d]$ covariance matrix $C_{\mathbf{R}}$ of an arbitrary rectangular region $\mathbf{R}$ within F then becomes our feature descriptor.

Each $C_{\mathbf{R}}$ can be computed efficiently using integral images [Viola and Jones, 2004]. We compute the sum of each feature dimension as well as the sum of the multiplication of every two feature dimensions. Given these first and second order integral image tensors, it can be shown that the covariance matrix of *any rectangular region* can be extracted in $O(d^2)$ time [Tuzel *et al.*, 2006]. Using the ground truth data from Sec. 5.2.3.1, we extract covariance descriptors of each training feature and store the associated feature label for training a classifier. However, the d-dimensional nonsingular covariance matrix descriptors cannot be

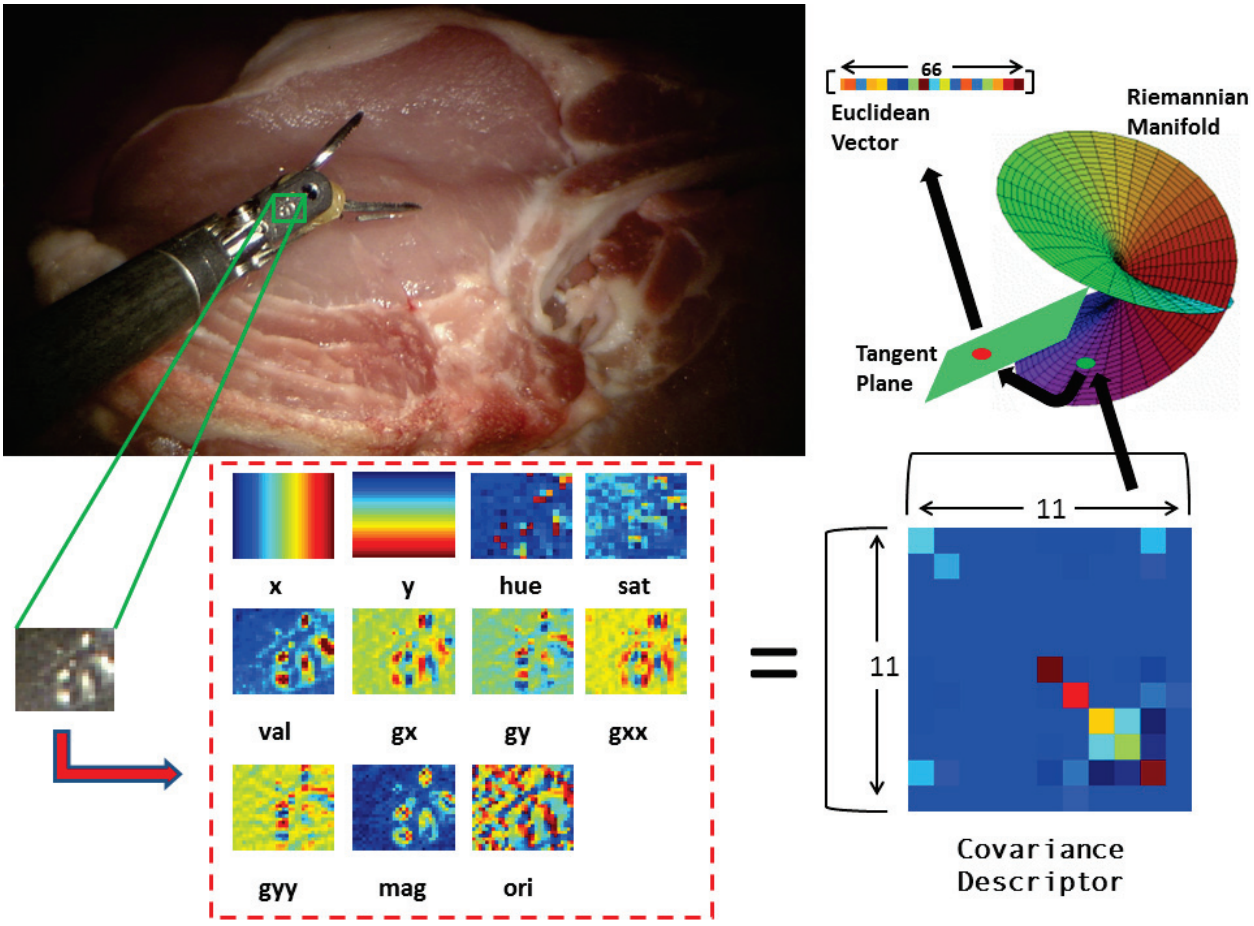


Figure 5.5: *Several independent features are combined compactly into a single feature descriptor*: We use 11 features overall (shown in the red dotted box), specifically the (x,y) locations, hue/saturation/value color measurements, first and second order image gradients, and gradient magnitude and orientation. A rectangular region (green box shown zoomed from the original image on the top) of the image is described by using the covariance matrix of these 11 features within that region, yielding an 11x11 symmetric matrix. In order to use this matrix as a descriptor with typical linear mathematical operations, we must map this matrix from it's natural Riemannian space to a vector space using Lie Algebra techniques (top-right), yielding a 66-dimensional vector space descriptor, described in more details in Sec. 5.2.3.2 and 5.2.3.3.

used as is to perform classification tasks directly because they do not lie on a vector space, but rather on a connected Riemannian manifold, and so the descriptors must be post-processed.

5.2.3.3 Post-Processing The Covariance Descriptors

An in-depth mathematical derivation for how to post-process the covariance descriptors to a vector space is shown in [Tuzel *et al.*, 2007]. Here we briefly summarize the procedure using the same notation. Symmetric positive definite matrices, of which our nonsingular covariance matrices belong, can be formulated as a connected Riemannian manifold [Pennec *et al.*, 2006]. A manifold is locally similar to a Euclidean space, and so every point on the manifold has a neighborhood in which a homeomorphism can be defined to map to a tangent vector space.

Our goal is to map our $[d \times d]$ dimensional matrices to a tangent space at some point on the manifold, which will transform the descriptors to a Euclidean multi-dimensional vector-space for use within our classifier. Given a matrix $\mathbf{X}$, we define the manifold-specific exponential mapping at the point $\mathbf{Y}$ as:

$$\exp_{\mathbf{X}}(\mathbf{Y}) = \mathbf{X}^{\frac{1}{2}} \exp(\mathbf{X}^{-\frac{1}{2}} \mathbf{Y} \mathbf{X}^{-\frac{1}{2}}) \mathbf{X}^{\frac{1}{2}} \tag{5.2}$$

and similarly for the logarithmic mapping:

$$\log_{\mathbf{X}}(\mathbf{Y}) = \mathbf{X}^{\frac{1}{2}} \log(\mathbf{X}^{-\frac{1}{2}} \mathbf{Y} \mathbf{X}^{-\frac{1}{2}}) \mathbf{X}^{\frac{1}{2}} \tag{5.3}$$

In these formulations, exp and log are the ordinary matrix exponential and logarithmic operations. Finally, we define an orthogonal coordinate system at a tangent space with the vector operation. To obtain the vector-space coordinates at $\mathbf{X}$ for manifold point $\mathbf{Y}$, we perform the following operation:

$$\mathbf{vec}_{\mathbf{X}}(\mathbf{Y}) = upper(\mathbf{X}^{-\frac{1}{2}} \mathbf{Y} \mathbf{X}^{-\frac{1}{2}}) \tag{5.4}$$

where $upper$ extracts the vector form of the upper triangular part of the matrix. In the end, we are left with a vector space with dimensionality $q = \frac{d(d+1)}{2}$.

The manifold point at which we construct a Euclidean tangent space is the mean covariance matrix of the training data. To compute the mean matrix $\mu_{\mathbf{C_R}}$ in the Riemannian space, we minimize the sum of squared distances:

$$\mu_{\mathbf{C_R}} = \underset{\mathbf{Y} \in \mathbf{M}}{\operatorname{argmin}} \sum_{i=1}^{N} d^2(\mathbf{X}_i, \mathbf{Y}) \tag{5.5}$$

This can be computed using the following update rule in a gradient descent procedure:

$$\mu_{\mathbf{C_R}}^{t+1} = \exp_{\mu_{\mathbf{C_R}}^{t}}[\frac{1}{N}\sum_{i=1}^{N}\log_{\mu^t}(\mathbf{X_i})] \tag{5.6}$$

We use the logarithmic mapping of $\mathbf{Y}$ at $\mu_{\mathbf{C_R}}$ to obtain our final vectors as in [Tuzel *et al.*, 2007]. The training covariance matrix descriptors are mapped to this Euclidean space and are used to train the multi-class classifier, described next.

5.2.3.4 Randomized Tree Classification

There are many multi-class classifiers which may suit this problem, however runtime is an important factor in our choice of a learning algorithm. To this end, we adapt a method called *Randomized Trees* (RTs) [Lepetit and Fua, 2006] to perform our multi-class classification. In addition to providing feature labels, we would like to retrieve confidence values for the classification task which will be used to construct class-conditional likelihood images for each class. We previously performed a study of different feature descriptors (e.g., Scale-Invariant Feature Transforms (SIFT) [Lowe, 2004], Histograms-of-Oriented Gradients (HoG) [Dalal and Triggs, 2005], and the Covariance Descriptors previously described) paired with various classification algorithms (e.g., Support Vector Machines (SVM) [Cortes and Vapnik, 1995] and two variants on RTs, described next) in [Reiter *et al.*, 2012b]. In this work, we determined that the Covariance Descriptor paired with our adaptation of RTs achieves a sufficient level of accuracy and speed.

RTs naturally handle multi-class problems very efficiently while retaining an easy training procedure. The RT classifier Λ is made up of a series of L randomly-generated trees $\Lambda = [\gamma_1, \ldots, \gamma_L]$, each of depth m. Each tree γ_i, for $i \in 1, \ldots, L$, is a fully-balanced binary tree made up of internal nodes, each of which contains a simple, randomly-generated test that splits the space of data to be classified, and leaf nodes which contain estimates of the posterior distributions of the feature classes.

To train the tree, the training features are dropped down the tree, performing binary tests at each internal node until a leaf node is reached. Each leaf node contains a histogram of length equal to the number of feature classes b, which in our problem is seven (for each of the manually chosen landmarks shown in Fig. 5.4). The histogram at each leaf counts the number of times a feature with each class label reaches that node. At the end of the training session, the histogram counts are turned into probabilities by normalizing the counts at a particular node by the total number of hits at that node. A feature is then classified by dropping it down the trained tree, again until a leaf node is reached. At this point, the feature is assigned the probabilities of belonging to a feature class depending on the posterior distribution stored at the leaf from

training.

Because it's computationally infeasible to perform all possible tests of the feature, L and m should be chosen so as to cover the search space sufficiently and to best avoid random behavior. In this work, we used $L = 60$ trees each of depth $m = 11$. Although this approach has been shown to be very successful for matching image patches [Lepetit and Fua, 2006], traditionally the internal node tests are performed on a small patch of the luminance image by randomly selecting 2 pixel locations and performing a binary operation (less than, greater than) to determine which path to take to a child. In our problem, we are using feature descriptor vectors rather than image patches, and so we must adapt the node tests to suit our problem.

To this end, we use a similar approach to [Bosch *et al.*, 2007] in creating node tests for feature descriptor vectors. In our case, for each internal tree node we construct a random linear classifier h_i to feature vector $\mathbf{x}$ to split the data:

$$h_i = \begin{cases} \mathbf{n}^\mathbf{T}\mathbf{x} + z \leq 0 & \text{go to right child} \\ otherwise & \text{go to left child} \end{cases} \tag{5.7}$$

where $\mathbf{n}$ is a randomly generated vector of the same length as feature $\mathbf{x}$ with random values in the range $[-1,1]$ and $z \in [-1,1]$ is also randomly generated. This test allows for robust splitting of the data and is efficiently utilized as it is only a dot product, an addition, and a binary comparison per tree node. In this way, we train the tree with vectorized versions of the covariance descriptors and build up probability distributions at the leaf nodes. The resulting RT classifier Λ is our final multi-class classifier. The results from each tree γ_i are averaged across all L trees. However, we choose relatively small values for L and m for computation purposes, but the search space is still quite large given the appreciable amount of choices for randomly-created linear dot products at the internal tree nodes, and this leaves the training approach susceptible to randomness. To alleviate this, we modify the approach further.

5.2.3.5 Best Weighted Randomized Trees

We developed a method [Reiter *et al.*, 2012b] which is able to improve on the standard RT approach which we call *Best Weighted Randomized Trees*. The modification lies in two observations:

- Each tree γ_i is essentially a weak classifier, but some may work better than others, and we can weight them according to how well they behave on the training data

- Because of the inherent randomness of the algorithm and the large search space to be considered, we can show improvement by initially creating a *randomized tree bag* Ω of size $E \gg L$. This allows us to initially consider a larger space of trees, but we then evaluate each tree in Ω on the training data in order to select the best L trees for inclusion in the final classifier according to an error metric.

The latter point allows us to consider more of the parameter space when constructing the trees while retaining the computational efficiency of RTs by only selecting the best performers. In order to evaluate a particular tree on the training data, we look at the posterior probability distributions at the leaf nodes. First, we split the training data into *training* and *validation* sets (e.g., we typically use $\sim 70\%$ to train and the rest to validate). Next, all trees from the training set in Ω are trained as usual. Given a candidate trained tree $\tilde{\gamma}_i \in \Omega$, we drop each training sample from the validation set through $\tilde{\gamma}_i$ until a leaf node is reached. Given training feature $\mathbf{X_j}$ and feature classes $1,\ldots,b$, the posterior distribution at the leaf node contains b conditional probabilities $p_{\tilde{\gamma}_i}(y|\mathbf{X}_j)$ where $y \in 1,\ldots,b$. To evaluate the goodness of tree $\tilde{\gamma}_i$ on $\mathbf{X_j}$, we compare $p_{\tilde{\gamma}_i}(y_j|X_j)$ to the desired probability $\mathbf{1}$ of label y_j, and accumulate the root-mean squared (RMS) error of all training features $\mathbf{X_j}$ across all validation trees in Ω. The top L trees (according to the lowest RMS errors) are selected for the final classifier Λ. Our initial bag size for this work was $E = 125,000$ candidate tree classifiers, cut down to $L = 60$ trained trees for the final classifier.

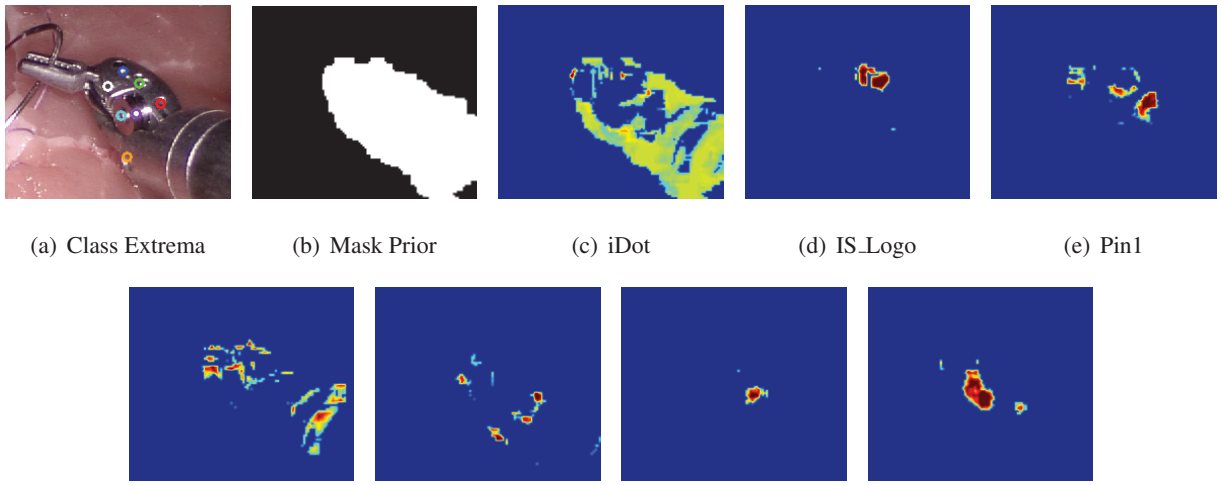
In addition to selecting the best trees in the bag, we use the error terms as weights on the trees. Rather than allowing each tree to contribute equally to the final averaged result, we weight each tree as *one-over-RMS* so that trees that label the validation training data better have a larger say in the final result than those which label the validation data worse. As such, for each $\gamma_i \in \Lambda$ we compute an associated weight w_i such that:

$$w_i = 1/rms_i \tag{5.8}$$

where rms_i is the accumulated RMS error of tree γ_i on the validation data. At the end, all weights w_i for $i \in 1,\ldots,L$ are normalized to sum to 1 and the final classifier result is a weighted average using these weights.

5.2.3.6 Feature Class Labeling

Given our trained classifier Λ, we detect features for each class label on a test image by computing dense covariance descriptors $\mathbf{C_R}$ (e.g., at many locations in the image) using the integral image approach for



(a) Class Extrema (b) Mask Prior (c) iDot (d) IS_Logo (e) Pin1

(f) Pin3 (g) Pin4 (h) Wheel (i) Wheel_Pin

Figure 5.6: Example likelihood images along with 6/7 successfully detected feature classes correctly located as extrema in (a). The *Pin3* (f) feature is incorrectly localized (white circle). The color-coding for the circles in (a) is: **Blue** for iDot, **Green** for IS_Logo (d), **Red** for Pin1 (e), **Orange** for Pin4 (g), **Purple** for Wheel (h), and **Cyan** for Wheel_Pin (i). A mask prior is shown in (b) which detects pixels on the metal part of the tool to reduce the number of pixels needing classification.

efficient extraction. Each $\mathbf{C_R}$ is mapped to a vector space using the mean covariance $\mu_{\mathbf{C_R}}$ of the training data as previously described, producing a Euclidean feature $\mathbf{c}_j$. We drop each $\mathbf{c}_j$ through the trees γ_t and average the probabilities at the obtained leaf nodes to get a final probability distribution p_L, representing the probability of $\mathbf{c}_j$ belonging to each of the L feature classes. This results in L class-probability images. To get the pixel locations, we perform non-maximal suppression in each class-probability image.

The reason we use the probabilities instead of the classification labels is that a classification of label l arises because its confidence is greater than all other $b-1$ classes in the classifier, however a confidence of 95% for one pixel location means more than a confidence of 51% for that same labeling at a different location. In this case, we would choose the pixel with the higher probability (even given they both have the same label), and for this reason we detect in probability space rather than in labeling space. Example detections and likelihoods resulting from the non-maximal suppression in the probability images using the metal mask prior (Fig. 5.6(b)) on the tool tip of an LND are shown in Figs. 5.6(a) and 5.6(c)-5.6(i), respectively.

5.2.3.7 Stereo Matching

Now that we have candidate pixel locations for each feature class, we stereo match the feature detections in the corresponding stereo camera using normalized cross-correlation checks along the epipolar line and triangulate the features to retrieve 3D locations. Using integral images of summations and squared-summations we can efficiently compute correlation windows along these epipoles. However, at this point we only have 3D point locations (in the camera's coordinate system) and associated feature labels, but we don't know with which tool each feature is associated. Next we describe the tool association procedure.

5.2.4 Patient-Side Manipulator (PSM) Association

At this point, we have class-labeled 3D feature locations, but with multiple tools in the scene it's unclear which feature is associated with which tool. Typically, the da Vinci® has three Patient-Side Manipulators (PSMs), only two of which are visible in the camera frame at any time. We label each manipulator as PSM_0, PSM_1, and PSM_2. For the purposes of this work we only consider two tools simultaneously, PSM_0 and PSM_1, and our goal is to associate feature detections with PSMs (module **D** in Fig. 5.2)

5.2.4.1 Pre-Processing The Marker Patterns

Each PSM has a marker pattern, M_0 and M_1, respectively, each in their *zero*-coordinate frame (e.g., the coordinate system before any kinematics are applied to the marker). Using the forward kinematics estimate from each PSM, we rotate the marker patterns to achieve the estimated orientations of each PSM. Note that we don't apply the full rigid-body transform from the forward kinematics because most of the error is in the position, and although the rotation isn't fully correct, it's typically close enough to provide the geometric constraints we require. This leaves us with:

$$\tilde{M}_0 = Rot_0(M_0) \tag{5.9}$$

$$\tilde{M}_1 = Rot_1(M_1) \tag{5.10}$$

where Rot_0 and Rot_1 are the $3x3$ rotation matrices from the full rigid-body transformations representing the forward kinematics for PSM_0 and PSM_1, respectively. Given $\tilde{M}_0$ and $\tilde{M}_1$, we compute 3D unit vectors between each of the rotated point locations within each marker. This yields $7x7$ 3D unit vectors in a $7x7x3$

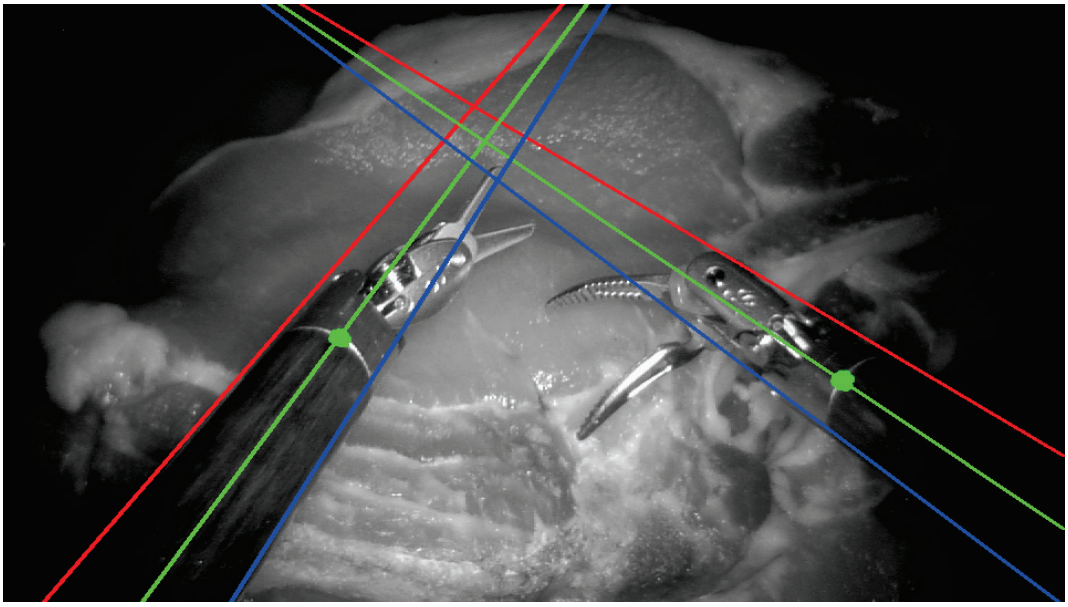


Figure 5.7: By extracting the boundary lines of the shaft (red and blue lines), the mid-line axis (green lines), and then the intersection location between the tool's shaft and the clevis (green dot), we can add shaft observations along with the feature observations to the fusion stage of the tracking system.

matrix for each rotated marker pattern. Additionally, we compute a $7x7$ distance matrix D_m between each marker location in its zero-coordinate frame.

5.2.4.2 Applying The Marker Geometry Constraints

Next, given N detected feature observations using the classification method described in Sec. 5.2.3, we compute both an NxN distance matrix between each 3D feature observation as well as an $NxNx3$ matrix of unit vectors, similar to those computed for the marker patterns using the kinematics estimates from the robot. Finally, we reject any feature observations which do not adhere to one of the pre-processed marker distance and rotation configurations according to the PSMs. Using empirically determined distance (e.g., $\sim 3-5$ mm) and orientation (e.g., $\sim 10^\circ - 20^\circ$) thresholds, we are able to determine which PSM is associated with each feature, allowing only one assignment per feature class to each PSM.

5.2.5 Shaft Extraction

As mentioned earlier, it is not guaranteed that there are enough shaft pixels visible to compute valid cylinder estimates, and so we use stereo vision to estimate the distance of the tool tip to the camera. If the algorithm determines that the tool's are situated far enough away from the camera so that the shaft is sufficiently visible, we use the shaft likelihood mask (from Sec: 5.2.2) to collect pixels in the image (potentially) belonging to one of the two tool's shafts (module **C** in Fig. 5.2). Assuming that each tool shaft is represented as a large, rectangular *blob*, using connected components and 2D statistical measures (e.g., aspect ratios, total pixel areas) we eliminate those areas of the image which are not likely to be one of the tool shafts.

Next, we fit 2D boundary lines to each candidate shaft blob, as shown with the blue and red lines in Fig. 5.7. Using projective geometry [Hartley and Zisserman, 2003] we fit a 3D cylinder to each pair of 2D lines, representing a single tool's shaft. Then, we locate the intersection point in the 2D image where the tool shaft meets the proximal clevis by moving along the cylinder axis mid-line from the edge of the image and locating the largest jump in gray-scale luminance values, representing where the black shaft meets the metal clevis (the green circles in Fig. 5.7). We then project a 3D ray through this 2D shaft/clevis pixel to intersect with the 3D cylinder and localize on the surface of the tool's shaft. Finally, we project this 3D surface location onto the axis mid-line of the shaft, representing a *rotationally-invariant* 3D feature on the shaft. This shaft feature is associated with its known marker location and is added to the fusion stage along with the feature classification detections (from Sec. 5.2.3).

The green lines in Fig. 5.7 represent the axis mid-lines of each tool shaft's cylinder representation. The intersection location is determined by moving along the (green) mid-lines from the edge of the image and locating the largest jump in change in gray-scale luminance, because the contrast of the black-shaft and metal-clevis will provide a very large jump in pixel value. The green dots along the green lines represent these shaft/clevis intersection locations, and each is projected to the associated 3D cylinder axis mid-lines, which each has a known, prior location on the shaft (and is invariant to the roll of the shaft) to be used as an additional input to the fusion stage, described next.

5.2.6 Fusion and Tracking

Because we cannot ensure that the features that we chose to detect are always visible on any given frame, we combine the robot kinematics with the vision estimates to provide our final articulated pose across time (module **E** in Fig. 5.2). The kinematics joint angles are typically available at a very high update rate,

although they may not be very accurate due to the error accumulation at each joint.

For surgical robots like the da Vinci®, it's important to keep the instrument insertion point (also termed *remote center*) stationary. This means that one part of the robotic arm holding the instrument does not move during the surgery (e.g., it is *passive*). The error of the end effector pose comes from both the error in zero calibration of the potentiometers at the joints and the error in the kinematics chain due to the link lengths. These are mostly static because the errors from the passive setup joints have more influence on the overall error as they are further up in the kinematic chain and have longer link lengths than the active joints. Therefore, if we can solve for this constant error *bias*, we can apply this to the raw kinematics of the active joints and end up with fairly accurate overall joint angle estimates. This *bias* essentially amounts to a rigid body pose adjustment at the stationary remote center. Although there is also error for the robotic arm holding the camera, when it does not move it is not necessary to include this in the error contributions.

To perform these adjustments on-line, we use an Extended Kalman Filter (EKF). The state variables for the EKF contain entries for the offset of the remote center and we assume that this is either fixed or slowly changing and so we can model it as a constant process. The observation model comes from our 3D point locations of our feature classes. We need at least 3 non-colinear points for the system to be fully observable. The measurement vector is:

$$\mathbf{y_3} = [x_1, y_1, z_1, \ldots, x_n, y_n, z_n]^T \tag{5.11}$$

The observation function which transforms state variables to observations is not linear, and so we need to provide the following Jacobians:

$$\mathbf{J}_1 = \frac{\partial \mathbf{p}^K}{\partial \mathbf{q}_I^K} \tag{5.12}$$

$$\mathbf{J}_2 = \frac{\partial \mathbf{p}^K}{\partial \mathbf{c}_I^K} \tag{5.13}$$

where $\mathbf{p}^K$ is a 3D point location in the kinematics remote center coordinate frame **KCS**, $\mathbf{q}_I^K$ is a unit quaternion rotation between the true instrument joint coordinate system **ICS** and the **KCS**, and $\mathbf{c}_I^K$ is the remote center location in the **KCS**. For more details, we refer the interested reader to [Zhao *et al.*, 2009b].

5.2.6.1 Handling Outliers

It is unlikely that any realistic solution to a computer vision problem does not contain outliers. We are mostly concerned with the image analysis as it is input to the fusion and tracking module (**E** in Fig. 5.2). To deal with this, we add an initial RANSAC [Fischler and Bolles, 1981] phase to gather a sufficient number of observations and perform a parametric fitting of the rigid transformation for the pose offset of the remote center. This is used to initialize the EKF and updates online as more temporal information is accumulated. We require a minimum of ~ 30 total inliers for a sufficient solution to begin the filtering procedure. The rigid body transformation offset is computed using the 3D correspondences between the class-labeled feature observations, done separately for each PSM after the PSM association stage described in Sec. 5.2.4, and the corresponding marker patterns after applying the forward kinematics estimates to the zero-coordinate frame locations for each tool. Because the remote center should not change over time, this pose offset will remain constant across the frames, and so by accumulating these point correspondences temporally, we are able to achieve a stable solution.

5.3 Experiments

We experimented on two types of datasets, both collected previously on a da Vinci® surgical robot: (1) porcine data (*in-vivo*), and (2) pork data (*ex-vivo*). The data which was used to test was specifically not included in the training collection procedure described in Sec. 5.2.3.1. After we collected and trained the seven feature classes using the $\sim 20,000$ training samples with our Best Weighted Randomized Trees approach (from Sec. 5.2.3.5), we applied the PSM association and geometric constraints method from Sec. 5.2.4 and finally the fusion and tracking stage from Sec. 5.2.6.

Overall, we experimented on 6 different video sequences, totaling 6876 frames (e.g., 458 seconds worth of video). Each video frame had two tools visible at all times. Across these video sequences, we experimented on three different types of da Vinci® tools, shown in Fig. 5.8. To demonstrate the strength of our system, we trained only on the Large Needle Driver (LND), shown on the left in Fig. 5.8, and tested on that same LND tool in addition to the Maryland Bipolar Forceps (MBF, middle) and Round Tip Scissors (RTS, right). The method works on these other tools because there are many shared parts across the tools, including the pins used to hold the clevis together and the **IS** logo in the center of the clevis. Even though the overall appearance of each tool is quite different, our results show that the method extends very well

Figure 5.8: Images of the three types of tools dealt with successfully in this work. We train only on the Large Needle Driver (left), and are able to track on all three, including the Maryland Bipolar Forceps (middle) and Round Tip Scissors (right).

to different tools given that the lower-level features are consistent. However, if newer tools are introduced which don't share these parts in common, more training data and feature classes must be considered and included in training the classifier discussed in Sec. 5.2.3.

We show 10 sample results in Fig. 5.9 from various test sequences. Rows 1-4 show *ex-vivo* pork results with different combinations of the LND, MBF, and RTS tools. Row 5 shows a porcine *in-vivo* sequence with an MBF on the left and an LND on the right. In Row 4 on the right-side, one tool is completely occluding the other tool's tip, however the EKF from the Fusion stage assists in predicting the correct configuration. For each, the red lines portray the *raw* kinematics estimates as given by the robot. The blue lines show the *fixed* kinematics after running our detection and tracking system. We show full video sequences for each of these as follows:

- Row 1 ("Seq. 1"), MBF (left) and LND (right): `http://www.youtube.com/watch?v=EWWQd-3zIT4`
- Row 2 ("Seq. 2"), RTS (left) and MBF (right): `http://www.youtube.com/watch?v=fT1wILqpY6w`
- Row 3 ("Seq. 3"), LND (left) and RTS (right): `http://www.youtube.com/watch?v=DD6ucZv5l2o`
- Row 4 ("Seq. 4"), MBF (left) and MBF (right): `http://www.youtube.com/watch?v=aGXR3BytGRs`
- Row 5 ("Seq. 5"), MBF (left) and LND (right): `http://www.youtube.com/watch?v=cNV12l_559g`

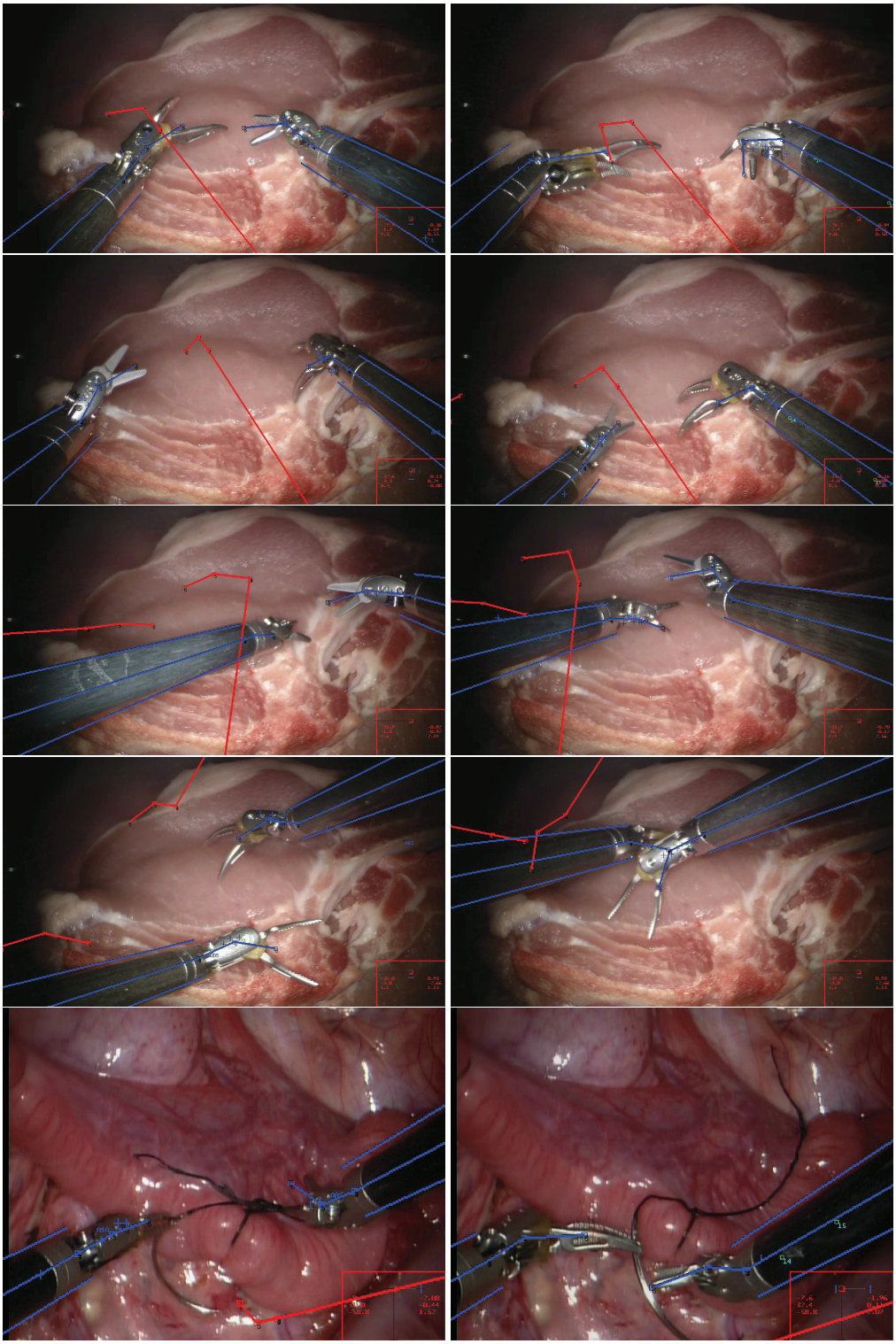


Figure 5.9: 10 sample results from various test sequences. Rows 1-4 show combinations of the 3 tools in Fig. 5.8 on the *ex-vivo* pork sequence. The last row shows a porcine *in-vivo* sequence. For each, the red lines show the *raw* kinematic estimates from the robot and the blue lines show the *fixed* kinematics after running our tracker. [**Row 1**] MBF (left) and LND (right) [**Row 2**] RTS (left) and MBF (right) [**Row 3**] LND (left)

One additional video result ("Seq. 6") is shown at `http://www.youtube.com/watch?v=TKiFQ3fKouM`. In these sequences, again the red lines represent the raw kinematics estimates from the robot, projected into the image frames. Notice the significant errors, where in some images the estimates are not visible at all, motivating the need for the algorithms presented in this chapter. The blue lines represent the *fixed* kinematics resulting from our tracking system. A visual inspection yields a fairly accurate correction of the kinematics overlaid on the tools.

Because joint-level ground truth for articulated tools is very difficult to collect accurately and on a large dataset, we evaluated the accuracy of our tracking system in the 2D image space. The left of Fig. 5.10 describes our evaluation scheme for our kinematics estimates. The dotted blue lines define an acceptable boundary for the camera-projection of the kinematics, where the green line is a perfect result. The right of 5.10 shows an example of an incorrect track on the right-most tool. Using this scheme, we manually inspect each frame of the test sequences, and resulted in a **97.81%** accuracy rate over the entire dataset.

Sequence	# Correct	Potential	% Correct
Seq. 1	1890	1946	97.12%
Seq. 2	2114	2182	96.88%
Seq. 3	1447	1476	98.04%
Seq. 4	1611	1648	97.75%
Seq. 5	4376	4431	98.76%
Seq. 6	1877	1930	97.25%
TOTAL	**13315**	**13613**	**97.81%**

Table 5.1: Tracking Accuracy Breakdowns

Table 5.1 shows a more detailed breakdown of our evaluation. Overall, we tested against 6 sequences, including both *ex-vivo* and *in-vivo* environments and all had two tools in the scene. The table shows the test sequence name in the first (leftmost) column, the number of tracks labeled as correct in the second column, the total possible number of detections in that sequence in the third column, and the final percent correct in the last (rightmost) column. Note that in any given frame, there may be 1 or 2 tools visible, and this is how we compute the numbers in the third column for the total *potential* number of tracks in that sequence.

Finally, the last row shows the total number of correct tracks detected as 13315 out of a total possible of 13613, yielding our final accuracy of 97.81% correct. Also note that the accuracy was very similar across the sequences, showing the consistency of the algorithm. Although the accuracy was evaluated in the 2D image space, we note that this does not completely represent the overall 3D accuracy as errors in depth may not be reflected in the perspective image projections.

5.3.1 Timing

The full tracking system runs at approximately $1.0 - 1.5$ secs/frame using full-sized stereo images (960x540 pixels). The stereo matching, PSM association, and fusion/EKF updates are negligible compared to the feature classification and detection, which takes up most of the processing time. This is dependent on the following factors: number of trees in Λ, depth of each tree γ_i, number of features used in the Region Covariance descriptor $\mathbf{C_R}$ (we use 11, but less could be used), and the quality of the initial segmentation providing the mask prior. However, by half-sizing the images we can achieve a faster frame-rate (0.6-0.8 secs/frame, an example of which is shown in Seq. 5) while achieving similar accuracy. Also, because we are solving for a remote center bias offset which remains constant over time, we can afford to process the frames at a slower-rate without affecting the overall accuracy of the tracking system. Finally, many stages

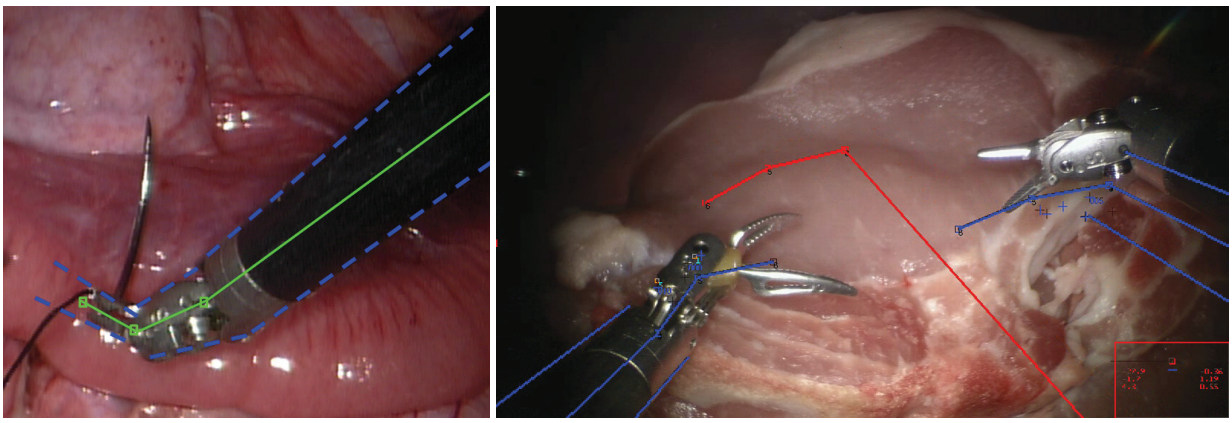


Figure 5.10: [**Left**] To evaluate our kinematics estimates, we evaluate in the image space because of the difficulty in collecting ground truth. The projected overlays must fall within the boundaries labeled as dotted blue lines here, where the green is a perfect detection. [**Right**] An example of an *incorrect* track on the right-most tool.

of the classification are parallelizable, and we are currently looking at implementing both the Covariance Descriptor and Randomized Trees on a GPU processor. Preliminary results on the covariance processing reduces the processing time of the feature tensors (Eq. 5.1) from $\sim$ 700ms to $\sim$ 100ms, and we believe we can reduce this further. We save for future work the GPU parallelization of the descriptor and classification procedures.

5.4 Discussions

5.4.1 Descriptor Window Size

There are many important choices to be made when implementing this tracking system. One such decision is in the size of the window to use when extracting covariance descriptors for classification throughout the image. The reason is that, during training, we use the best encompassing bounding box around each feature, and the descriptors are well tuned to representing the entire feature. When applying the classifier, if the window is too small or too large, the descriptors won't capture the features well. To alleviate this, we use prior knowledge of the 3D sizes of the features to guide computation of the optimal window size. Using the stereo vision approach which determines if the shaft is visible enough to extract (from Sec. 5.2.2) and estimating that the features are $\sim 3x3mm$ in size, we can automatically determine the optimal window size in the image *dynamically* on each frame. To further reduce errors, at every pixel location that we evaluate, we extract a bounding box which is both full and half-sized according to this automatically determined window size to account for the smaller features (e.g., the pins). This improves the overall feature detection system.

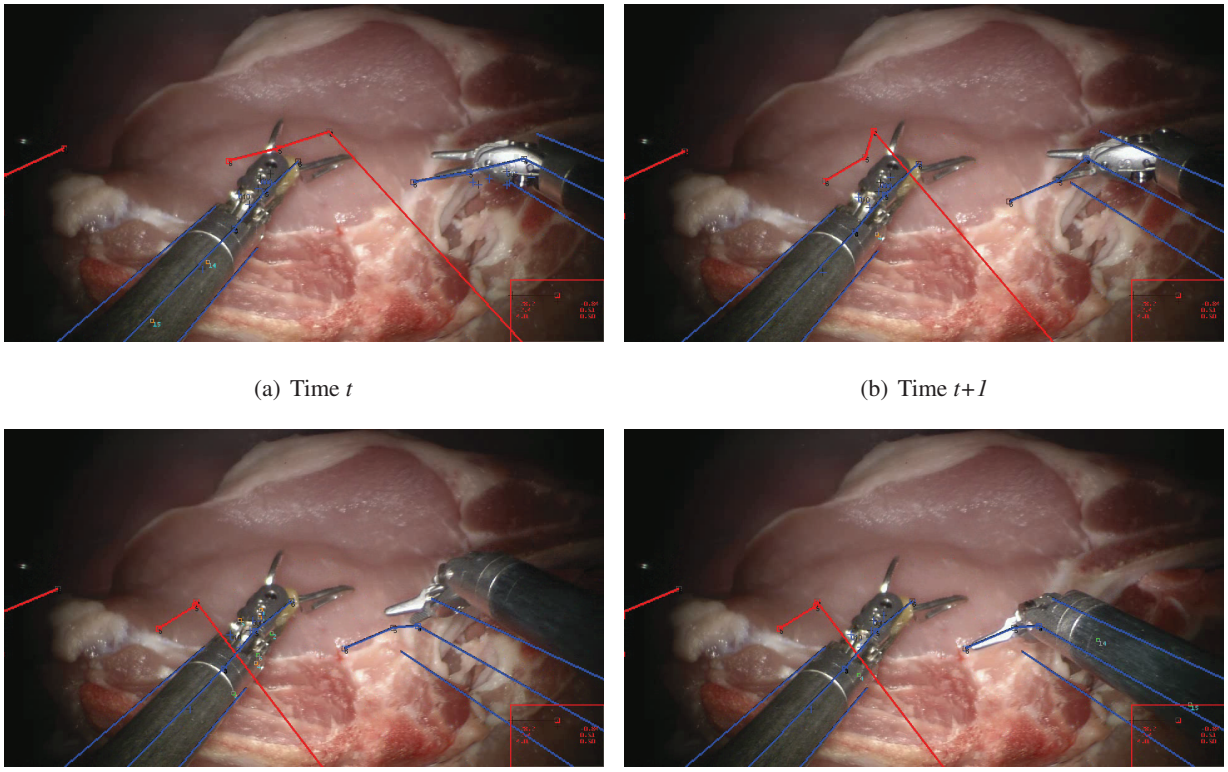
5.4.2 Kinematics Latency

Upon further inspection of the errors encountered during evaluation on the test sequences, we found that most of the incorrect *fixed/tracked* kinematic configurations are due to a latency in the raw kinematics which causes the video and raw kinematics to be out-of-sync from time-to-time. This situation is shown more precisely in Fig. 5.11. We determined this by noticing that, for the individual frames which had incorrect projections (according to our scheme described in Sec: 5.3), the result would jump immediately to a correct configuration instead of getting lost completely, and the previous incorrect projection was in the location and configuration that the upcoming projection would eventually reach. Therefore, by logging the test data more precisely so that the video and kinematics are more in sync with each other, we would expect

our accuracy to increase even further. However, in practice on a live system this kinematic latency does not exist, and future *in-vivo* live experiments should demonstrate this.

5.4.3 Hybrid Approach

Finally, we wish to mention that the majority of tool tracking approaches in the literature work by estimating the cylinder of the shaft which is visible in the scene [Voros *et al.*, 2007; Doignon *et al.*, 2006]. However, as we previously discussed, surgeons tend to work quite zoomed in, making this cylinder-fitting procedure



(a) Time *t* (b) Time *t+1*

(c) Time *t+2* (d) Time *t+3*

Figure 5.11: Example of kinematic latency (right tool): often the kinematics and video get out-of-sync with each other. Most of our errors are due to this fact, manifesting in the situation shown here. In (a), both tools are tracked well. Then, in (b) and (c), the kinematics and video become out-of-sync and the right tool becomes inaccurately tracked. However, in (d), the tools are tracked successfully again. Looking at the blue configuration in (c), which is essentially the same as the correct one immediately following in (d), suggests this latency is the source of our errors. These four frames are consecutive to each other in order.

very difficult, if not impossible, due to the limited number of visible shaft pixels. The remaining minority approaches work by analyzing the tip of the tool using features [Burschka *et al.*, 2004; Reiter and Allen, 2010], however these will fail when the tool tip is too far away to be seen well by the camera. Our approach is advantageous in that it dynamically decides which of these two approaches is optimal at any given time, and often uses both simultaneously to best the track the tool over longer periods of time. Also, by using the pixel labeling method described in Sec. 5.2.2, we are able to tell more accurately when parts of the tool are occluded. For example, if the metal tool tip is occluded then the pixel labeling won't label the incorrect pixels from the occluder as metal, and we will avoid less false positives, and similarly for the shaft.

5.5 Live In-Vivo Experiments

To further demonstrate the abilities of this tracking algorithm, we ran the tracker live during a two-hour long porcine surgical demonstration using a variety of tools and realistic scenarios to fully gather the strengths and weaknesses. The robotic tools and endoscope were inserted into the abdominal area to inspect the kidney and surrounding anatomy. We first tested on the LND tools, as a baseline, and four sample snapshots of this sequence are shown in Fig. 5.12. We also tested on a novel tool which the system had never seen before, which shares a similar distal clevis to the MBF, but has a fairly different overall appearance. Two screen-shots of successful tracking for this tool are shown in Fig. 5.13.

5.5.1 Realistic Occlusions

During a real surgery, occlusions are almost guaranteed to occur on the tool. Often, the occluder comes in the form of blood which gets stuck on the tool (affecting the appearance), or tissue which gets stuck on the tool and blocks different portions of the tool. We tested several of these scenarios in different situations. Fig. 5.14 shows four different occluding scenarios. On the top-left corner, the image shows an example of a piece of suturing tape which got stuck on the tool tip, however enough features are still detected to enable the tracking. In the top-right and middle-left images, tissue which was previously cut got stuck on the right-most MBF tool, and this surely affects the appearance of the tool. The feature detection and classification is still able to find enough landmarks to feed into the tracker to recover the pose. The middle-right image shows both tissue stuck on the right-most MBF tool and also dried blood on left-most LND tool. Similarly, the bottom-left and bottom-right images show blue-dye which was placed on the left-most LND tool (and

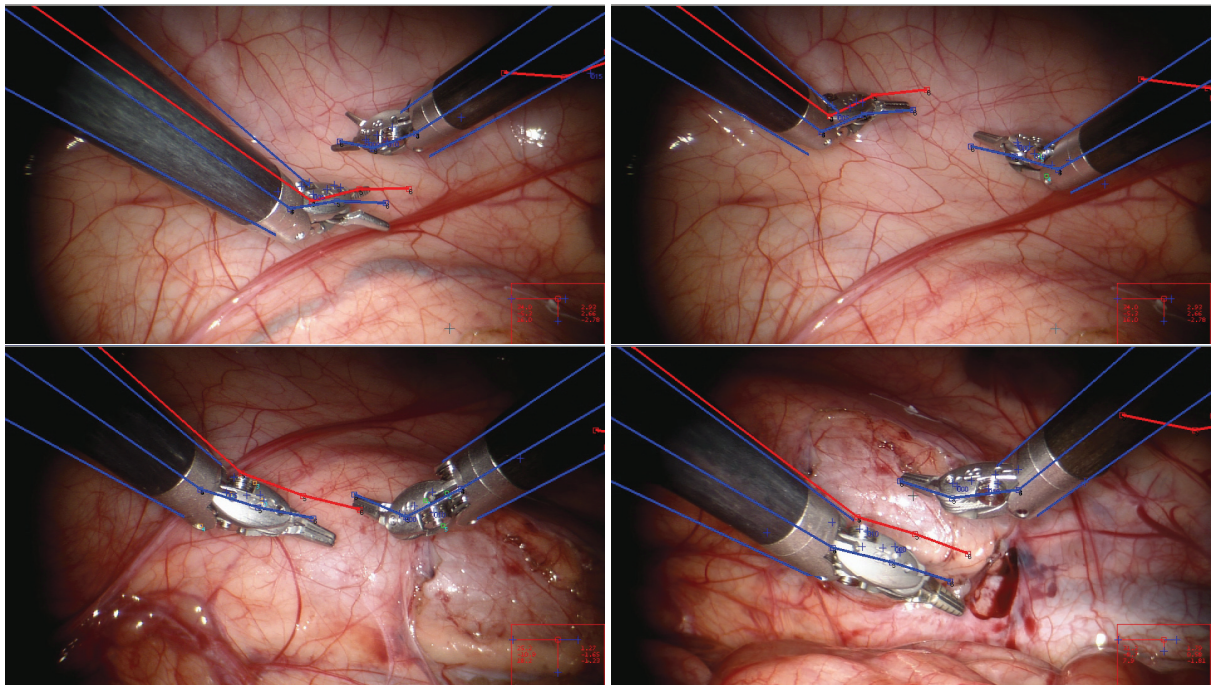


Figure 5.12: Snapshots of our live in-vivo porcine experiment demonstrating two LND tools in a realistic surgical scenario. In this situation, the tracker was run *live* rather than on off-line video data.

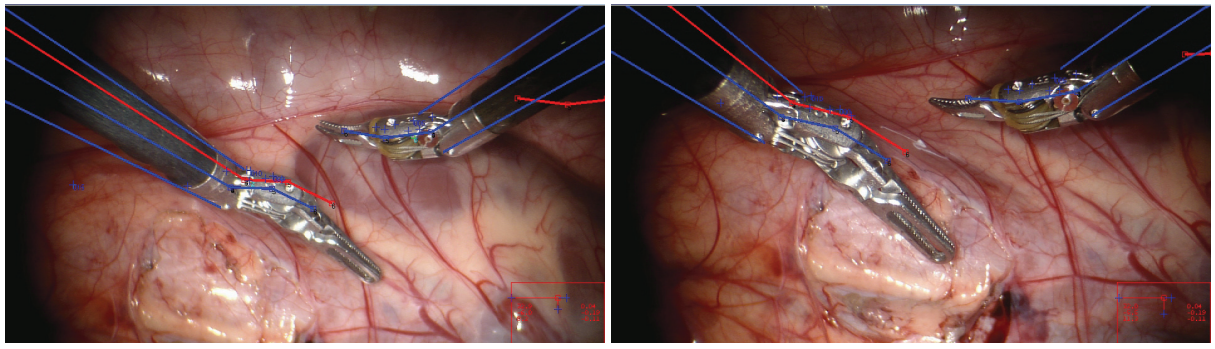


Figure 5.13: Running the tracker *live* on a novel tool (on the **left**, never seen before by the system) in an in-vivo porcine surgical demonstration. The novel tool shares a similar distal clevis to the MBF tool, however it has a very different overall appearance.

dried blood on the right-most MBF tool), and through all of these realistic occlusion scenarios the feature detection was robust enough to detect the landmarks for the tool tracker to succeed for long periods of time.

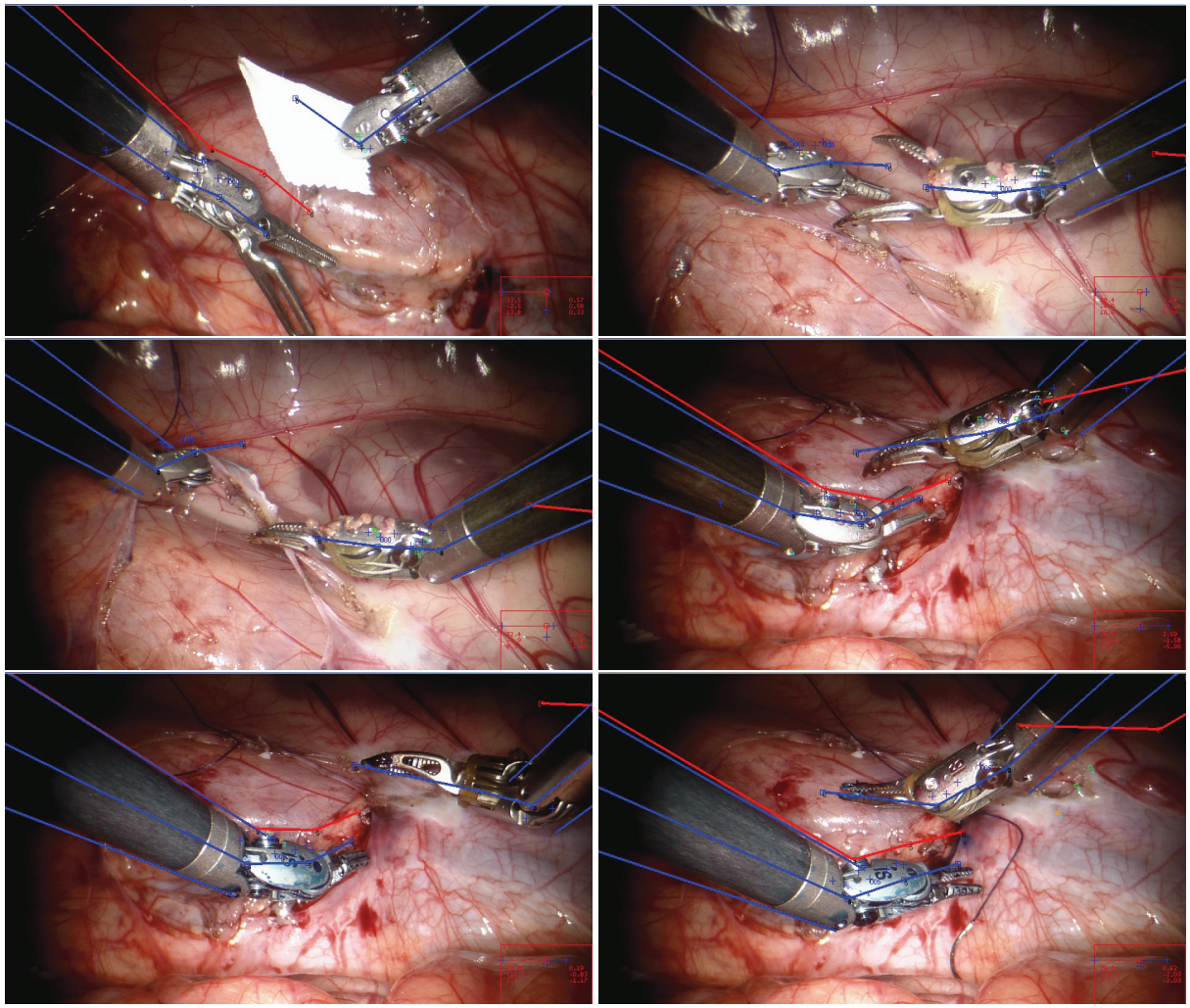


Figure 5.14: Various degrees of occlusion during the live in-vivo porcine demonstration. The top-left image shows a piece of suturing tape which got stuck on the tip of the right-most LND tool, however features are still detected and the tracker persists through this occlusion scenario. On the top-right and middle-left, tissue which was previously cut got stuck on the right-most MBF tool, affecting the appearance of the tool. However, enough features were detected for successful tracking. On the middle-right, the right MBF still has tissue stuck on the clevis, and the left LND tool has dried blood. Similarly, on the bottom-left and bottom-right images, we placed blue-dye on the left-most LND tool (and the bottom-right image shows dried blood also on the right-most MBF tool), and through all of these scenarios, the feature detection was robust enough to detect the important landmarks for the tracking to persist.

5.5.2 Lessons Learned

It's important to understand that the success or failure of this tracking algorithm lies in its ability to detect the features which were previously learned in realistic operating conditions. The main goal of the live demo was to determine if this feature detection scheme was sufficient for localizing these important landmarks across various conditions which will occur during real surgeries. We tested through the occlusions mentioned above as well as smoke resulting from electro-cautery stimulation and squirting liquid (e.g., saline), which is used to clean off tools and different portions of the environment. Through all of these scenarios, enough features were always detected to facilitate tracking.

The features do not need to be detected on every frame, due to the fusion and prediction parts of the algorithm. What may vary from scenario to scenario is the initial time needed to acquire the tool. Using the RANSAC method described in Sec. 5.2.6.1, which accumulates 3D observations of feature detections over time, we are able to acquire track of the tool faster if we get more features over smaller periods of time. However, if the tool is not showing visible features often enough, it may take several seconds (or longer) for the tool to be acquired, due to the visibility of the landmarks on which the system is trained. Although we did not do a rigorous study on this aspect of the algorithm (due to time constraints and difficulty in collecting ground truth), we did qualitatively notice different acquisition times corresponding to different degrees of difficulty. However, in every single case we tested the tool was eventually always acquired and successfully tracked.

5.6 Conclusions

This chapter has presented a tool detection and tracking framework which is capable of tracking multiple types of tools and multiple tools simultaneously. The algorithm was demonstrated on the da Vinci® surgical robot, however it may be extended to other types of surgical robots. We showed high accuracy and long tracking times across different kinds of environments (*ex-vivo* and *in-vivo*). By learning low-level features using a multi-class classifier, we showed how different degrees of visibility for each feature can be overcome. We also showed that a hybrid approach of using both the shaft and features on the tool tip is advantageous over either of these methods alone. Using knowledge of the distance of the tool we can dynamically adapt to different levels of information into a common fusion framework. Finally, by fusing vision and kinematics, we can account for missed observations over time. We were able to show the tracker working live and in-

vivo, in realistic surgical scenarios where we considered occlusions and different types of tools as well as operating conditions with varying degrees of difficulty.

Part II

Continuum Shape Estimation

Chapter 6

IREP: Single-Segment Configuration Estimation

6.1 Motivations

The work discussed thus far relates to improving visual situational awareness (via tool tracking) for minimally-invasive laparoscopic surgical procedures. The da Vinci® is a robotized example of a laparoscopic tool, but it still suffers from the limitations previously mentioned. Although the wristed instruments increase flexibility at the distal end of the operations, multiple incisions are still required and the relatively large size precludes operations in small areas.

The introduction of SPA [Romanelli and Earle, 2009] and NOTES [Lehman *et al.*, 2008] procedures require robot designs such as the IREP [Xu *et al.*, 2009; Simaan *et al.*, 2012]. In this chapter, we describe our work in estimating the flexible configuration states of the snake robot arms of the IREP using vision [Reiter *et al.*, 2011]. This would be used for a closed-loop control system to perform *automated* tasks, such as suturing, needle biopsies, or blood suctioning. The accuracy of the algorithm must be high for the tasks to be realistically completed, and so our approach attempts to achieve sub-degree precision. We begin by demonstrating the algorithm on a single-segment, manually-actuated continuum robot arm. In Ch. 7 we show how to extend this to multiple segments on a mechanically-actuated, 3-segment continuum robot.

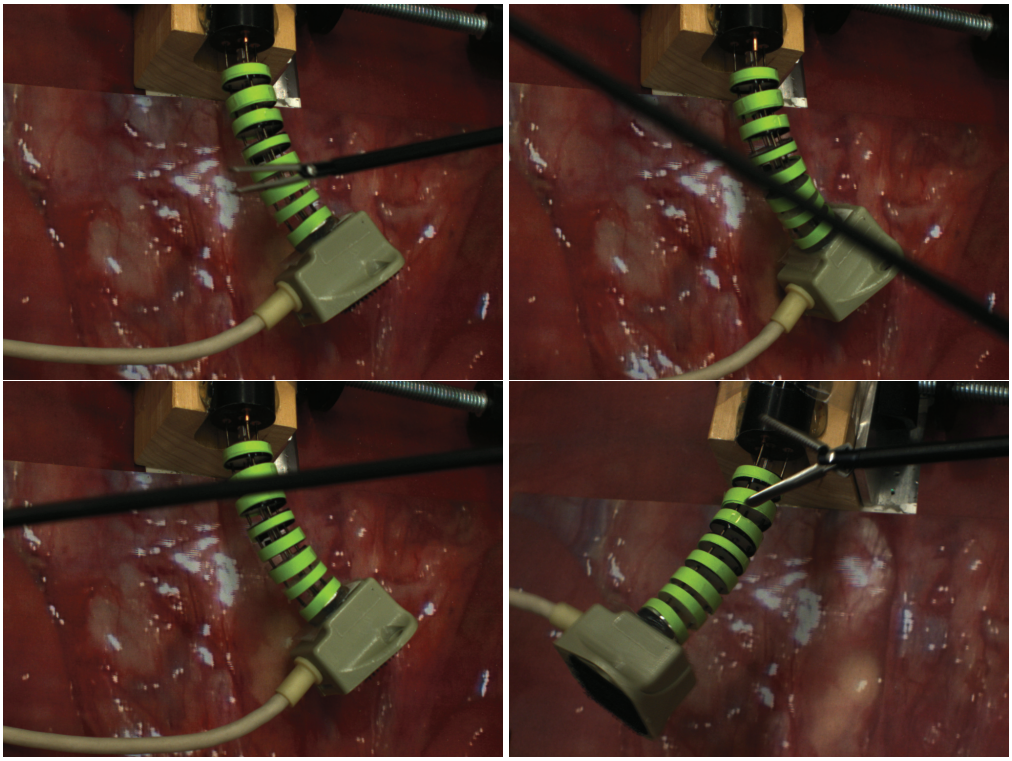


Figure 6.1: A stereo camera system views a single segment of a continuum snake arm in order to learn a mapping of the configuration angles from visual feature descriptors. The algorithm was tested in the presence of a realistic occluder in the form of a laparoscopic tool being manipulated between the snake and the camera.

6.2 Overview

This work proposes a method to estimate the configuration of a single-segment continuum robot using a visual feature descriptor that is extracted from a stereo camera system and mapped to the robot's configuration angles. Our image segmentation and descriptor extraction methods are shown to be robust to partial occlusions. We present these results by manipulating a standard laparoscopic tool in the viewing frustrum, providing different levels of partial occlusions in a realistic fashion. Although the algorithm extends to any continuum robot design, we have in mind the IREP surgical robot [Xu *et al.*, 2009], shown on the right of Fig. 6.2. Our method uses training samples to interpolate a manifold, which is parameterized by the configuration angles of the continuum segment. This compact representation of the appearance of the robot's configuration allows us to estimate unknown configurations by extracting the feature descriptor and indexing

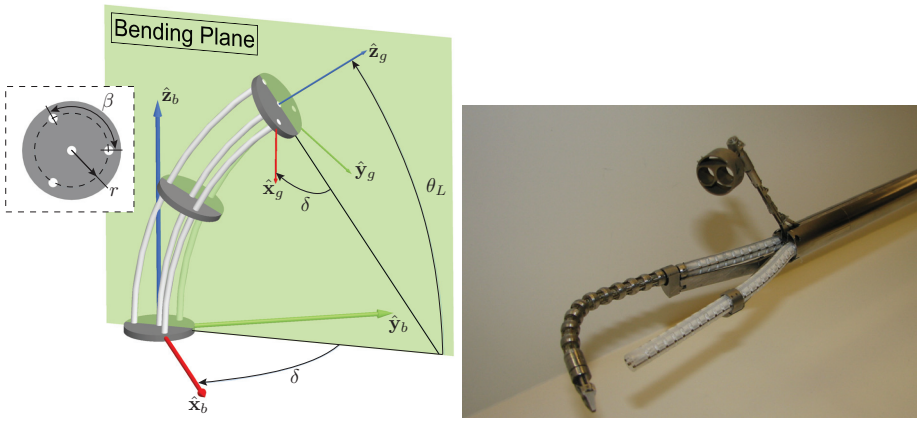


Figure 6.2: [**Left**] Structure and kinematic nomenclature for a single-segment continuum robot. [**Right**] Early prototype of IREP with one continuum arm constructed along with the camera housing for the stereo camera actuation system.

into the manifold to determine the best angles which may have produced that descriptor. The proposed algorithm uses a variation of a well-known feature descriptor [Belongie *et al.*, 2002] to provide the sensitivity needed to accurately capture small changes in the configuration of a continuum robot. Although the segment bends in a circular shape, camera perspective effects make measuring the image of the circle (viewed as an ellipse) difficult when the movement is out-of-plane, and the descriptor encodes this information in an alternative and robust fashion. The algorithm relies on the assumption that consecutive configurations are strongly correlated and nearby in the feature descriptor's space. Once the training section is completed, the algorithm's output can directly feed into the controller in [Bajo *et al.*, 2011] for closed-loop feedback. We tested on robot movements in all 3-dimensions and are able to recover configuration angles in the range of 1° of accuracy.

6.2.1 Modeling of the Continuum Segment

Several designs of continuum robots that bend in a circular shape have been proposed [Webster III and Jones, 2010]. This section briefly presents the kinematics of the particular design [Simaan *et al.*, 2004] used to validate the work proposed below. The multi-backbone single-segment robot shown on the left of Fig. 6.2 is constructed of one centrally located passive primary backbone, and three radially actuated secondary backbones with pitch radius r and separation angle β. By controlling the lengths of the secondary backbones,

the segment can be moved throughout the workspace defined by the kinematics. The pose of the end disk of the continuum robot can be completely described by the generalized coordinates, termed configuration space, by

$$\psi = [\theta_L, \delta]^{\mathrm{T}} \tag{6.1}$$

where θ_L and δ define respectively the angle tangent to the central backbone at the end disk, and the plane in which the segment bends. The orientation of the end disk is given by the following sequence of rotations:

$$\mathbf{R} = \mathbf{R_z}\mathbf{R_y}\mathbf{R_z^T} \tag{6.2}$$

where $\mathbf{R_z} = \mathrm{e}^{-\delta[\mathbf{e_3}\times]}$, $\mathbf{R_y} = \mathrm{e}^{(\theta_0-\theta_L)[\mathbf{e_2}\times]}$, denote the exponential forms for these rotations, $\mathbf{e}_j$ denote the canonical basis unit vectors for $\mathbb{R}^3$, $[\mathbf{n}\times]$ designates the skew-symmetric cross product matrix of vector $\mathbf{n}$, and $\theta_0 = \pi/2$. The configurations variables θ_L and δ can be obtained from (7.1) as:

$$\theta_L = \theta_0 - \mathrm{atan2}\left(\sqrt{R_{13}^2 + R_{23}^2}, R_{33}\right) \tag{6.3}$$

$$\delta = -\mathrm{atan2}\left(R_{23}, R_{13}\right) \tag{6.4}$$

where R_{ij} are the entries of rotation matrix $\mathbf{R}$[1]. Because the lengths of the individual segments are constant, the full pose can then be computed using these rotation angles.

6.2.2 Learning Method

The problem of estimating the pose of an object by learning the appearance has been studied previously. Murase and Nayar [Murase and Nayar, 1995] collected a set of images by sampling the workspace of an object's configuration and compressing to a low-dimensional eigen-subspace. This builds a continuous appearance manifold for which queries can be interpolated for unknown poses. This particular approach is extremely sensitive because the appearance is represented at the pixel level, and spatial variations within the image may present a challenge. Occlusions present an issue as well, which is extremely common in surgical environments. This parametric eigenspace representation is the main motivation for the contributions of our ideas, with the additions of spatial invariance due to the use of a feature descriptor and a reduced dimensionality of the image representation.

[1] We use the atan2 notation such that: $\theta = \mathrm{atan2}\left(\sin(\theta), \cos(\theta)\right)$.

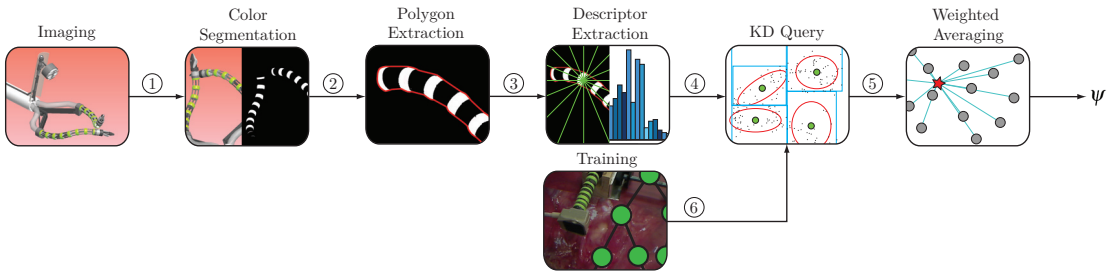
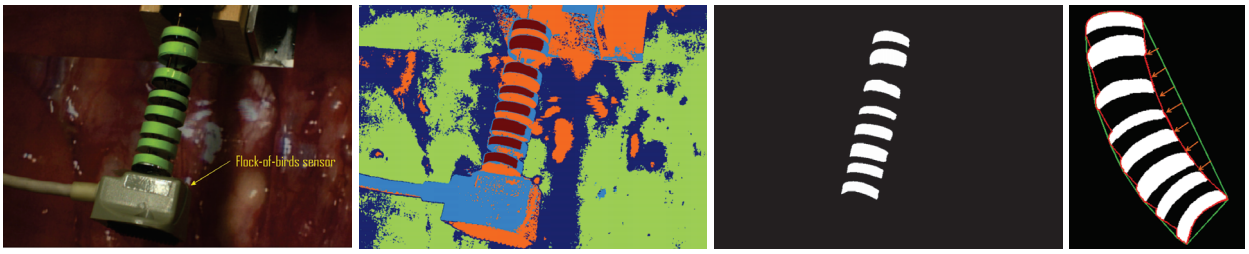


Figure 6.3: Algorithm flow for the visual pose estimation algorithm. ① Stereo Images, ② Segmented Components, ③ Polygonal Points, ④ *eigen-features*, ⑤ b closest interpolated poses, ⑥ Initial training set $\mathbf{A}$. Further details in text.

Vision offers a low-cost and safe solution to physical measurements in a surgical environment. Because the configuration of a single-segment continuum robot can be completely described by configuration angles, it would be a powerful argument to do so accurately with cameras alone. Fig. 6.3 shows the full algorithm flow of our method for describing the configuration of a bendable arm using vision. The method relies on *learning* mappings of configurations to feature descriptors. The descriptors must be stable enough so that nearby points in feature space represent similar configurations. By discretely sampling a continuous space of configurations, we can interpolate a continuous feature descriptor manifold, which is parameterized by the configuration angles, and then accurately and efficiently estimate the unknown configuration of the arm by indexing into the manifold and matching to the nearest neighbors of the descriptors in the manifold and performing a weighted average from the known configuration angles nearby.

In addition to learning, another positive aspect to our algorithm lies in its ability to estimate the configuration using only partial data due to visual occlusions. Often during surgery, overlapping tools or excess liquids may temporarily occlude parts of the continuum arm. Our algorithm robustly and successfully persists despite partial occlusions.

For the remainder of this chapter we will refer to the static, base part of the snake segment as the *proximal* portion of the arm, and the movable endpoint of the segment as the *distal* portion. Note that several of these segments are often combined together to form a full robotic snake arm for maximal dexterity, and this algorithm proposes a solution to estimate the configuration of each single segment at a time, which can then be combined in the end to capture the full pose of the robot arm. We first need to train our system to map known poses to feature descriptors.



(a) Color Image (b) Cluster Labels (c) Segmented Image (d) Contour

Figure 6.4: Color segmentation is performed by using a single color frame [**A**] and a known set of color clusters (5 in our experiments) to label pixels in an image according to the closest color cluster [**B**] in CIELAB color space. The marker pixels fall out cleanly from the learning procedure, labeled as red in **B**. This results in a binary labeling [**C**] according to this cluster of pixels on the markers we wish to analyze. The contour being extracted [**D**] represents a concave polygon. We achieve this by starting with the convex hull, shown in green, and then moving in the points along the contour towards the closest binary-labeled pixel location. The result is shown in red, and this gives a good approximation to the best encompassing contour around the segment.

6.2.2.1 Ground Truth Collection

We fixed an *Ascension Technology Flock-of-Birds* 3D tracking device to the distal end of the snake segment (see Fig. 6.4(a)). This is a magnetic tracker capable of providing 3D positions and orientations at approximately $144Hz$. Positional and orientation accuracy are 1.8mm and 0.5° RMS, respectively. The sensor provides a 3D position $\mathbf{p}$ and a 3D orthonormal rotation matrix $\mathbf{R}$, which we convert to ψ angles according to (7.3) and (7.4). This ground truth is sufficient to describe the moving configurations of the snake segment.

6.2.2.2 Snake Segmentation

Color Segmentation A snake arm with 8 vertebrae is color coded with lime markers. The length of the segment is 61mm, and each vertebrae disk has a height of 3.5mm with a diameter of 14mm. This color was chosen so as to stand out from typical medical imagery, which is more red by nature. It is not unrealistic to place these kinds of fiducial markers on surgical robots to simplify these types of detection tasks during surgeries [Wei *et al.*, 1997]. We begin with a single frame to perform an unsupervised *k-means clustering* of colors. We choose the CIELAB color space, which consists of a luminosity layer **L**, chromanicity layer-

a (which indicates where colors fall along the red-green axis), and chromanicity layer-*b* (which indicates where colors fall along the blue-yellow axis). The color information is actually in the *a* and *b* layers, and so the clustering is performed using only those 2 components. We found this to be a more robust representation to capture and cluster color pixels than the typical RGB or HSV color spaces which are commonly used. The *a* and *b* layers are normalized by the **L** layer to provide robustness to lighting changes (section 6.3.2).

We make the assumption that in surgical imagery, a finite number of representative colors are present at any given time. By using a single frame, we specify the number of color clusters we expect to show up. For purposes of our experiments, we use 5 clusters according to our environment. An example is shown in Fig. 6.4, using sample surgical imagery as the background. The original image from the right camera is shown in 6.4(a), with a superimposed arrow showing where the flock of birds is located. In Fig. 6.4(b) we show the result of the k-means clustering using the *ab* components. Here, pixels are labeled according to the closest of the 5 learned clusters. The colored markers stand out quite cleanly, and are labeled as red pixels. Note that no other pixels in the image are red except for on the markers. This learning stage is performed only once, in the beginning, and we select the cluster label corresponding to the markers to label subsequent images. Then, for any given image, we first convert from RGB to CIELAB. Next, for each pixel, we find the closest cluster according to the k-means result and label as 255 if it's the label we previously selected and 0 otherwise. A sample segmented result is shown in Fig. 6.4(c).

Contour Extraction Now that we have a binary image, we wish to extract the best encompassing contour about the segmented region. The challenge here lies in the gaps between the separated vertebrae. So as to not be specific to our hardware, we want our algorithm to be applicable in the case of a continuous bendable segment without gaps. We begin by computing the convex hull around the binary pixel locations, shown in green in Fig. 6.4(d). However, as the segment bends the boundary actually forms a *concave polygon*. One half of the segment will be convex, and the hull will be correct there, but the concave portion will be off. The convex hull represents a set of vertices of line segments which form the boundary of the polygon, however we want a denser sampling of this boundary. To achieve this, we recursively interpolate linearly between successive points in the hull, producing control points along the boundary. Then we find the nearest pixel locations to these control points that originally created the hull (the binary labeled pixels), thereby *moving* the contour locations of the concave areas *inwards* towards the true boundary of the object. The result is shown as the red contour in Fig. 6.4(d). The orange arrows show the need for this, as contour locations in the concave area need significant adjustment. The closest points can be found either using a linear search

or a kd-tree. Both the dimensionality and the number of points is small, and so these searches are not computationally intensive.

As a final step, to get an even and dense sampling of the contour, we take the points that make up the contour and apply the Bresenham line algorithm [Bresenham, 1965] to consecutive points, assuming local linearity between close points. To augment this contour, we also compute a binary edge map from the segmented image (Fig. 6.4(c)) using a Sobel operator. This yields points that are interior to the polygon on the boundaries of the markers. We add in these points with the contour locations to build the descriptor, described next. This can be helpful in the case of occlusions, where the contour alone may get deformed, and the edges help diminish the effect of the deformation. Conversely, the edges are not sufficient to fill-in the gaps between the separated markers.

6.2.2.3 Descriptor Extraction

Next we seek to build a descriptor to represent the 3D pose of the object. The scheme is to compute a feature descriptor on each of the stereo images separately and then combine them together to form a single,

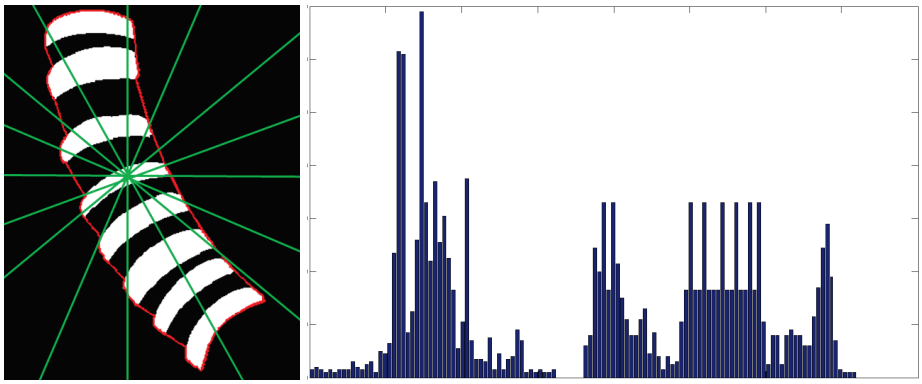


Figure 6.5: [**Left**] The descriptor (for a single camera) is constructed by taking points along the contour (red) of the snake segment and computing the angle of each point with respect to the center of the object. The angular bins are depicted as green lines emanating from the center of the segmented region. [**Right**] A densely-binned histogram counts the number of points falling within each angular bin. The drawing on the left is an example showing 16 bins for space considerations, however in our experiments we looked at 72 (every 5 degrees), 120 (every 3 degrees), and 360 (every 1 degree) bins per image. The x-axis for the histogram represents angular bins and the y-axis represents hit-counts.

composite stereo feature vector. Although we are not explicitly performing 3D reconstruction, 3D shape information is being encoded because we have two separate views of the object in a stereo setup, and ambiguous movements due to out-of-plane perspective effects in one camera can be captured by the other camera. We take the location of the center of the segmented region, and build a 1D histogram of the angles of each point along the contour with respect to the center of the object. Fig. 6.5 describes this conceptually where the left image shows the angular bins radiating from the center location, displayed as green lines. The bins count the number of points in each angular range and build a histogram, as shown on the right. The histogram should be densely binned so that small changes in shape are captured and the descriptor is sufficiently sensitive, yet not overly noisy. We experimented with bins of size 5° (72 bins), 3° (120 bins), and 1° (360 bins), ultimately choosing the 120-bin case.

The intuition behind this representation is that as the segment bends, the shape redistributes the points along the contour in unique ways. By counting the number of points in each bin, we can analyze the redistribution of these points as the shape changes. It can be thought of as the same total number of points in the bins across the frames, yet redistributed into different distributions within the histogram to capture the shape changes. The example shown on the left in Fig. 6.5 has 16 bins for drawing purposes, however in practice we extract much denser bin sizes. This descriptor can be thought of as a *Shape Contexts* descriptor [Belongie *et al.*, 2002], but without distance information and much finer-scaled in terms of bin size.

6.2.2.4 Training

Pre-Processing The training phase consists of mapping known ground truth configuration angles to feature descriptors. First we must collect the raw data from the sensors. Using the magnetic sensor and a stereo camera system, we collect pairs of stereo images $S_i = \{I_{Li}, I_{Ri}\}$ and associated $\psi_i = \{\theta_{Li}, \delta_i\}$ measurements, where $i = 1, ..., N$ for N discrete training samples.

Each stereo pair S_i is mapped to a feature descriptor $\tilde{H}_i = [H_{Li}^T, H_{Ri}^T]$ to represent the shape of the segment on that frame. Here, H_{Li} is the feature descriptor extracted from the i^{th} left image, and similarly H_{Ri} from the right frame, represented as a single composite feature vector $\tilde{H}_i \in \mathbb{R}^{m \times 1}$.

This gives us an initial training set $\mathbf{A} = \{(\psi_1, \tilde{H}_1), \ldots, (\psi_N, \tilde{H}_N)\}$. However, each $\tilde{H}_i$ is quite high-dimensional, and also may be sparse. Murase and Nayar [Murase and Nayar, 1995] show that a compact representation of an object's appearance is sufficient for accurate pose estimation by creating a *parametric eigenspace* to represent the appearance. In their case, the images themselves are projected to a lower-

dimensional eigen-subspace. For our algorithm, we similarly compute the principal components of the full training set of $\tilde{H}_i$ samples, noting that this is done in feature descriptor space rather than on the original images. In our experiments, we found that we can reduce the dimensionality quite significantly while still recovering a large percentage of the variance.

The principal component analysis (PCA) over $\tilde{H}_i$ yields a set of orthonormal basis vectors $\{e_1, \ldots, e_m\} \in \mathbb{R}^{m\times 1}$. We choose $k < m$ of these eigenvectors to capture a sufficient percentage of the variance of the original feature descriptor training dataset, giving a linear transformation matrix $E = [e_1, \ldots, e_k] \in \mathbb{R}^{m\times k}$, where the columns of E are each of the k basis vectors e representing the top k eigenvalues of the PCA. We project the original training feature descriptor samples in $\mathbf{A}$ to the eigen-subspace:

$$L_i = E^T(\tilde{H}_i - c) \tag{6.5}$$

where c is the mean feature descriptor over all $\tilde{H}_i$, and $L_i \in \mathbb{R}^{k\times 1}$. We will call the L_i samples *eigen-features*. This gives a final training dataset $\hat{\mathbf{A}} = \{(\psi_1, L_1), \ldots, (\psi_N, L_N)\}$.

Parametric Manifold Interpolation We assume that consecutive features are very highly correlated, and so their projections into the eigen-subspace are close together. Our experiments are consistent with this assumption (see section 6.3). The discrete points L_i describe a smooth parametric manifold represented in eigen-subspace as:

$$G(\psi) = L \tag{6.6}$$

Depending on the number of degrees-of-freedom (DOFs) represented by ψ, the shape of G will vary. For example, if $\psi \in \mathbb{R}^1$, then G represents a curve in k-dimensional space. Similarly, if $\psi \in \mathbb{R}^2$, G is a surface, and so on. Using our representation, we obtain a *parametric manifold* which is a surface. The discrete samples are used to interpolate this manifold by performing spline interpolations of ψ to L.

A spline is a piecewise polynomial function which can be defined on an N-dimensional space to produce a single function value at each N-dimensional point. In order to fit points in $\mathbb{R}^2$ to points in $\mathbb{R}^k$, as our manifold describes, we must perform k 2-dimensional spline fits to each of the eigen-feature's output dimensions separately. We use a thin-plate smoothing spline interpolation of the configuration angles $\psi_{i=1,\ldots,N}$ to the eigen-features $L_{i=1,\ldots,N} \in \mathbb{R}^{k\times 1}$. This gives us k sets of spline interpolant coefficients $G_j(\psi_i) = L_{ij}$, where $j = 1, \ldots, k$ corresponds to the j^{th} eigenvector's projection dimension. We then use these interpolant coefficients to *resample* the manifold at a higher density over ψ's workspace. This yields interpolated descriptors

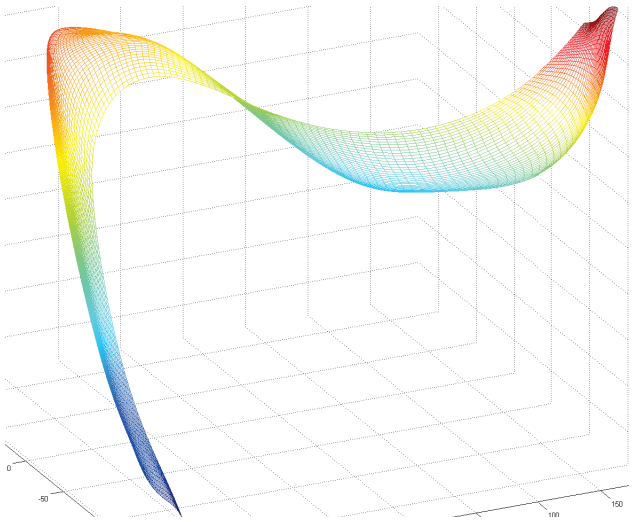


Figure 6.6: The interpolated parametric manifold over the 2-DOF pose angles ψ, yielding a surface in the feature descriptors eigen-subspace, called *eigen-features*. For purposes of drawing, we only show the first 3 dimensions of the eigen projections. However in our experiments the eigen-subspace $\in \mathbb{R}^k$, where k is the number of principal components we choose from the PCA step.

which smoothly describe these estimated configuration angles.

An example of this manifold is shown in Fig. 6.6 for the first 3 dimensions of the eigen-features, corresponding to the 3 largest eigenvalues. The discrete training samples that were originally collected (and then projected via PCA) exist as points on (or close to) this manifold. Finally, the resampled manifold points are stored in a *kd-tree* for the querying stage, described next.

6.2.2.5 Querying

Now that we have constructed a densely-sampled manifold and stored it within an efficient look-up data structure, we can query unknown configurations in a very straightforward manner. For any test frame T, we extract the feature $\tilde{H}_T$ and using the eigenvector projection matrix E we project $\tilde{H}_T$ using (6.5) to obtain the test eigen-feature L_T. We then use the kd-tree from the interpolated manifold to find the b closest points on the manifold to L_T. In our experiments $b = 3$, and each of the b matches provides Euclidean distances d_l in feature space, where $l = 1, \ldots, b$. We use these distances to compute weights, representing the contributions each will make to the final pose estimate, based on proximity in feature space. The weights w_l are computed

as:

$$w_l = \frac{1/d_l}{\sum_p \frac{1}{d_p}} \tag{6.7}$$

The weights contribute as the inverse of the distances in feature space, so that closer points contribute more than further points. The denominator in (6.7) is provided so that the weights sum to 1. Then a weighted average of the pose angles that created those interpolated eigen-feature matches provides the final configuration estimate for $\tilde{H}_T$:

$$\psi_T = \sum_{l=1}^{b} w_l \psi_l \tag{6.8}$$

6.3 Experiments & Results

6.3.1 Accuracy of Pose Estimation

First we evaluate the main part of the algorithm, which is to estimate the configuration angles ψ using feature descriptors. We used a stereo camera system composed of two 1024x768 resolution color Point Grey Research Dragonfly2 [Poi,] cameras mounted on a tripod. During our experiments, the cameras and the snake base remained static. The snake segment was color coded with lime-green markers and we created a background using printouts of photos from a laparoscopic procedure. The segment was manually actuated and the ground truth positions and rotations were collected by interfacing with the Flock-of-Birds sensor.

Training Data First, we collected stereo image pairs and associated ground truth rotation measurements. We choose the first frame of the image stream to construct the CIELAB color clusters. Then we manually select the label which corresponds to our marker color. For each subsequent image, we classify each pixel as previously described. At the same time, we convert all rotation matrices to ψ configuration angles according to (7.3) and (7.4).

These training samples are used to form the initial training dataset $\mathbf{A}$ and then the modified training dataset $\hat{\mathbf{A}}$ according to the method described in 6.2.2.4. The manifold shown in Fig. 6.6 is the result of this training procedure, again only showing the first 3-dimensions according to the top 3 eigenvalues from the PCA projections. For our experiments, we collected 2296 training samples.

We experimented with different dimensionalities of the feature descriptors extracted from the individual images. We analyzed bin sizes of 1, 3, and 5 degrees, corresponding to histogram dimensionalities of 360,

120, and 72, respectively. Note that in these cases, the composite stereo descriptor $\tilde{H}$ is twice as long, specifically 720, 240, and 144, respectively. We also experimented with different degrees of dimensionality-reduction in the PCA step in order to test the effect of the loss of dimensions to the overall accuracy. In our experiments, for each of the bin sizes, we looked at different percentages of variance recovery: 65%, 85%, 90% and 95%.

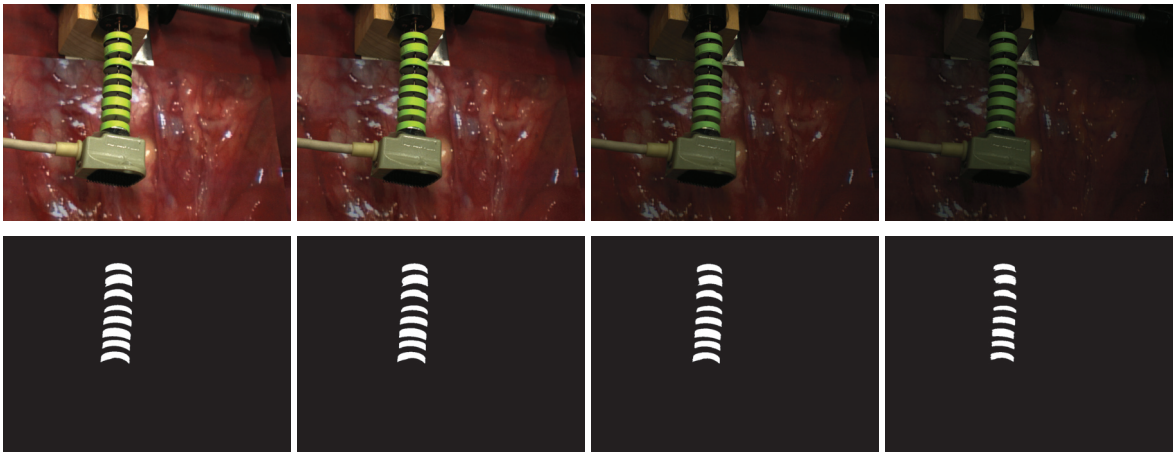
Testing Data Testing data is collected in the same way as the training data. For each test measurement T, we create the feature descriptor $\tilde{H}_T$ and then the eigen-feature L_T using the projection matrix E obtained from the training data. No test data was used in the creation of the manifold.

Pose Accuracy Table 6.1 shows results of some of the dimensionality reductions for each of the bin sizes of our feature descriptors. Because we use a kd-tree for feature matching on the eigen-features, we want to reduce this dimensionality for faster matching. In our experiments, we found the best combination of accuracy and run-time efficiency occurring with 120-bin feature descriptors reduced down to 90% variance, resulting in stereo eigen-features L which reside $\in \mathbb{R}^{16}$.

It's important to note that more bins isn't necessarily better. At a certain point discriminability becomes noise, and so we chose 120-bins over 360-bins as the best trade-off, both outperforming the 72-bin case quite consistently. The 120-bin case reduced to 16-dimensions in stereo yielded a configuration accuracy with errors of $[\varepsilon_\delta = 0.98^\circ, \varepsilon_{\theta_L} = 1.28^\circ]$ over 806 test samples. With both angles combined together, the overall median pose error was 1.16°.

In our experiments, we could not sample a full 360° workspace because the snake segment was fixed to a table so that the base could not move. The workspace of our experiments consisted of about 3/4 the entire workspace of the segment, cycling the δ angle 180° through its range and θ approximately 70° for each within-plane rotation. We wanted to ensure that out-of-plane rotations from the imaging plane are sufficiently captured by means of the stereo system. Even though the segment bends in a circular arc, due to perspective effects of camera imaging systems, out-of-plane rotations are viewed as conic sections rather than circles. Often these can be difficult to recover by ellipse-fitting methods, and so our descriptor mapping approach is ideal to avoid these difficult shape-fitting problems.

Occlusions We also tested our algorithm against occlusions by manipulating a common laparoscopic tool near the segment, occluding the view from the cameras in a realistic fashion. Although accuracy degrades slightly, we are able to achieve errors of $[\varepsilon_\delta = 1.04^\circ, \varepsilon_{\theta_L} = 2.06^\circ]$, for a combined accuracy of 1.46°. Fig. 6.1 shows sample images displaying the types of occlusions that were dealt with.



(a) Bright (b) Normal (c) Dark (d) Darker

Figure 6.7: The robustness of the color segmentation method is tested under different lighting conditions. [**Top row**] Color images from which pixels are classified to produce binary images [**bottom row**]. We use a single image to train with, under normal lighting conditions [**B**]. Subsequently we classify pixels under brighter [**A**] and darker [**C**, **D**] lighting.

Table 6.1: Dimensionality Reduction

N-bins	**% Var**	**Dims**	**% Var**	**Dims**	**% Var**	**Dims**
72	65	2	85	5	95	19
120	65	2	85	8	95	39
360	65	3	85	39	95	200

6.3.2 Robustness of Color Segmentation

Next we evaluate the robustness of the color clustering technique by testing how well it performs under different lighting conditions. To achieve robustness to lighting, we normalize the pixels in CIELAB color space before clustering or labeling. The luminosity layer contains information about the overall brightness of the scene. Therefore, we divide each pixel in the chromanicity a and b layers by the maximum value in the **L** layer, both for training and classification. The result is shown in Fig. 6.7. Here we train with only a single image, under *normal* lighting conditions (Fig. 6.7(b)). We then classify pixels as previously discussed on the images in the top row, which results in binary images shown in the bottom row. Fig. 6.7(a) shows the light-level turned brighter, and Figs. 6.7(c) and 6.7(d) show the light-level turned dimmer. The

segmentation doesn't suffer significantly.

However, there is a limit to how bright or dim the light level can get before the classification fails, and the method can be broken. To stress test this technique, we varied the luminosity over a range of $108/255$ levels, yet the chromanicity-a and b layers only changed, on average, by about 4 levels, or 1.6% of its range.

6.4 Discussions

6.4.1 Dimensionality

The dimensionality of the original feature descriptor is something to be thought about, as well as the dimensionality reduction step. Histograms can pose a danger in that small changes in angle may be filtered out due to the natural binning effect of histograms. To counter this we use a smaller bin size, resulting in a high-dimensional feature vector. PCA is meant to rid the feature of the sparseness while retaining the useful information. Our feature descriptor can be reduced significantly while capturing a very large percent of the variance (240 down to 16).

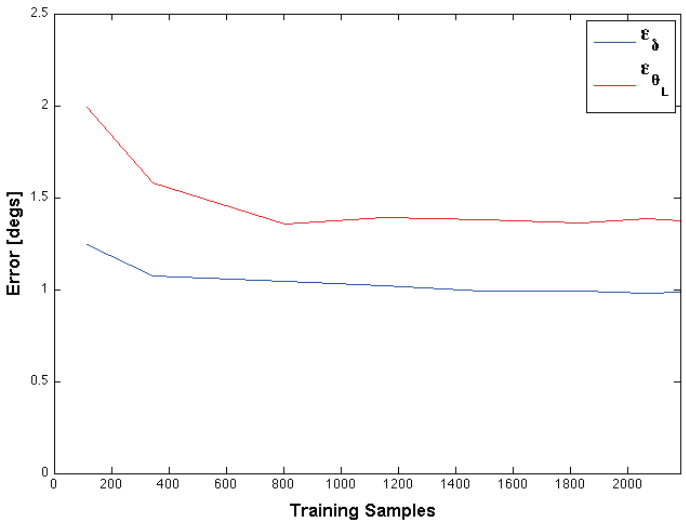


Figure 6.8: The results of randomly permuting different percentages of the training data to interpolate the parametric manifold and the effect of this on accuracy. To avoid outliers, the trial for each percentage was done 3 times and the average error was taken as the result.

6.4.2 Training Set Size

Another important aspect is in the number of training samples required. Depending on how densely the workspace is sampled, the accuracy of the manifold will vary. If we collect a very dense set of configuration measurements, the problem is reduced to a nearest neighbor matching problem. A strength of the manifold representation is that if a small number of training samples is used, the underlying system may still be captured. Fig. 6.8 shows this observation where we randomly permuted the training samples and selected different percentages to interpolate the manifold and determined the effect on the accuracy. Each trial was done 3 times to avoid outliers, and the average error is shown on the y-axis in degrees. We show the error in δ and θ_L separately as the blue and red lines, respectively. Note that even when we only use 15% of the training data (345/2296), we still obtain reasonable results, proving the strengths of the manifold method. In this case, a nearest neighbor approach would be insufficient and the interpolation is required.

6.4.3 Generalizing The Method

This algorithm is applied to the continuum robot described in this chapter to estimate the configuration angles ψ because these angles completely describe the configuration space of a single segment of the snake arm. However, it's important to note that this method can be extended to interpolate the manifold parametrically using any DOFs which apply to a robot.

To describe this idea further, suppose that instead of estimating the bending plane angle δ and within-plane angle θ_L, we wanted to track the 3D position of the endpoint of the segment. In this case the manifold would be parametric in 3-DOFs since we are using the endpoint position rather than ψ rotations, and this would result in a manifold volume rather than a surface. We ran this experiment using the same data as described in section 6.3, but using positions from the Flock-of-Birds rather than rotations, and we obtained a median positional accuracy of 0.97mm. In other words, our method can be used to accurately measure different aspects of the robot, depending on the application. In this way, the approach is general enough to different domains and robots, and extensible to different feature descriptors as well.

Chapter 7

IREP: Multi-Segment Configuration Estimation

7.1 Motivations

In this chapter, we extend the previous algorithm (Ch. 6), which estimates the configuration of a single-segment continuum robot by interpolating a smooth functional mapping from robot configurations to features using a look-up table. The extension to multiple segments is time-consuming because a high-dimensional nearest-neighbor lookup (with *many* samples) is required.

The contribution of this work [Reiter *et al.*, 2012d] is a method for estimating the configuration of a multi-segment continuum robot arm using a similar visual learning algorithm. Off-line ground truth robot configuration angles are mapped to stereo visual feature descriptors through a smoothly-varying parameterized feature manifold. Radial Basis Functions [Buhmann, 2003] are used to interpolate this high-dimensional feature manifold to *directly* estimate the configuration angles from the features, avoiding any memory-intensive look-up tables in the reverse direction as we did in the single-segment case. The coupling effects of the independently bending segments are accounted for by expressing the orientation of each segment with respect to a static base coordinate frame. The application of a smoothing operation to the shape descriptor increases the stability and repeatability of the pose estimation through the manifold.

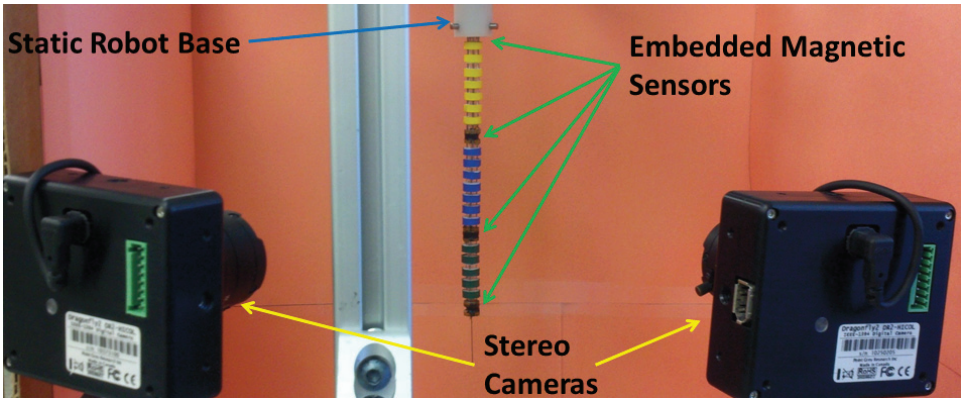


Figure 7.1: The stereo cameras [**Bottom**, yellow arrows] are used to extract stereo feature descriptors and map them to configuration angles. Our training algorithm uses ground truth from embedded magnetic sensors [**Top-Right**, green arrows] which provide relative rotations between segments. Our system expresses the configuration of each snake segment as a rotation to the static robot base [**Top-Left**, blue arrow] so that each segment can be formed in the same coordinate frame and be computed individually, regardless of coupling effects due to the kinematic chain of the proximal segments above it.

7.2 Modeling of the Multi-Segment Continuum Robot

A single-segment snake arm is constructed of one centrally located passive primary backbone, and three radially placed secondary backbones with pitch radius r and separation angle β, as shown on the left of Fig. 7.2. The segment is moved through the workspace by controlling the lengths of the secondary backbones. The pose of the end disk of segment $k \geq 1$ can be completely described by the generalized coordinates:

$$\psi_k = [\theta_{L_k}, \delta_k]^T \tag{7.1}$$

where θ_{L_k} and δ_k define respectively the angle tangent to the central backbone at the end disk and the plane in which the segment bends. The orientation of the end disk is given by the following sequence of rotations:

$$\mathbf{R_k^{k-1}} = \mathbf{R_Z R_Y R_Z^T} \tag{7.2}$$

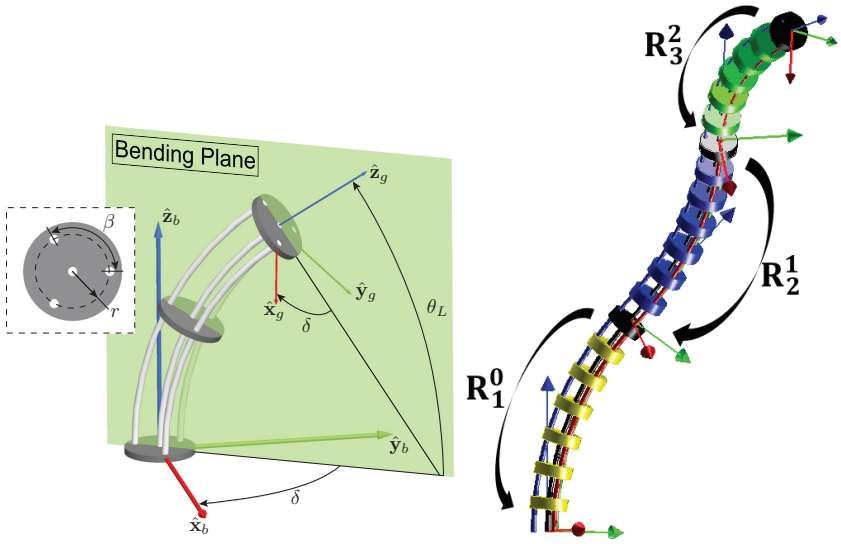


Figure 7.2: [**Left**] Structure and kinematic nomenclature for a single segment continuum robot. [**Right**] Drawing of our 3-segment continuum robot, displaying coordinate systems and the rotations which represent the configurations.

where $\mathbf{R_z} = Rot(-\delta_k, \hat{\mathbf{z}})$, $\mathbf{R_y} = Rot(\theta_0 - \theta_{L_k}, \hat{\mathbf{y}})$, and operator $Rot(\phi, \mathbf{u})$ returns a rotation matrix of angle ϕ about axis $\mathbf{u}$ and $\theta_0 = \pi/2$. The configuration variables θ_{L_k} and δ_k can be obtained from (7.2) as[1]:

$$\theta_{L_k} = \theta_0 - \text{atan2}\left(\sqrt{R_{13}^2 + R_{23}^2}, R_{33}\right) \tag{7.3}$$

$$\delta_k = -\text{atan2}\left(R_{23}, R_{13}\right) \tag{7.4}$$

where R_{ij} are the entries of rotation matrix $\mathbf{R}_k^{k-1}$.

To augment our approach to multiple segments (Fig. 7.2, right), we use a global representation of each segment, regardless of its kinematic chain. As such, we seek a rotation $\mathbf{R}_k^0$ which maps the end disk of any segment k to the static base $\{0\}$ of the robot. To achieve this, we chain together rotations of segments, proximally, all the way up to the base coordinate frame. For example, given sequentially-ordered segments, it's assumed that the rotation matrix $\mathbf{R}_k^0$ is formed by composition:

$$\mathbf{R}_k^0 = \mathbf{R}_1^0 \mathbf{R}_2^1 \ldots \mathbf{R}_k^{k-1} \tag{7.5}$$

In this way, we can express every segment of the snake arm in a common coordinate frame and train with these mappings all together. A 3D rotation matrix represents 3 independent DOFs, and so we want a 3-vector

[1] We use the atan2 notation such that: $\theta = \text{atan2}\left(\sin(\theta), \cos(\theta)\right)$.

to represent this transform. We chose to use **ZYX** *Euler angles*, $\mathbf{e}_k = [\alpha_{e_k}, \beta_{e_k}, \gamma_{e_k}]$, s.t.:

$$\alpha_{e_k} = \arcsin(\frac{R_{21}}{\eta}) \tag{7.6}$$

$$\beta_{e_k} = \arcsin(-R_{31}) \tag{7.7}$$

$$\gamma_{e_k} = \arcsin(\frac{R_{32}}{\eta}) \tag{7.8}$$

where η is a normalization factor:

$$\eta = \sqrt{R_{11}^2 + R_{21}^2} \tag{7.9}$$

In this formulation, α_{e_k} is a rotation about the z-axis, β_{e_k} is a rotation about the y-axis, and γ_{e_k} is a rotation about the x-axis. Note that other rotation representations may also be used in place of this and easily substituted into our approach. We use $\mathbf{e}_k$ to represent the 3-vector Euler angles.

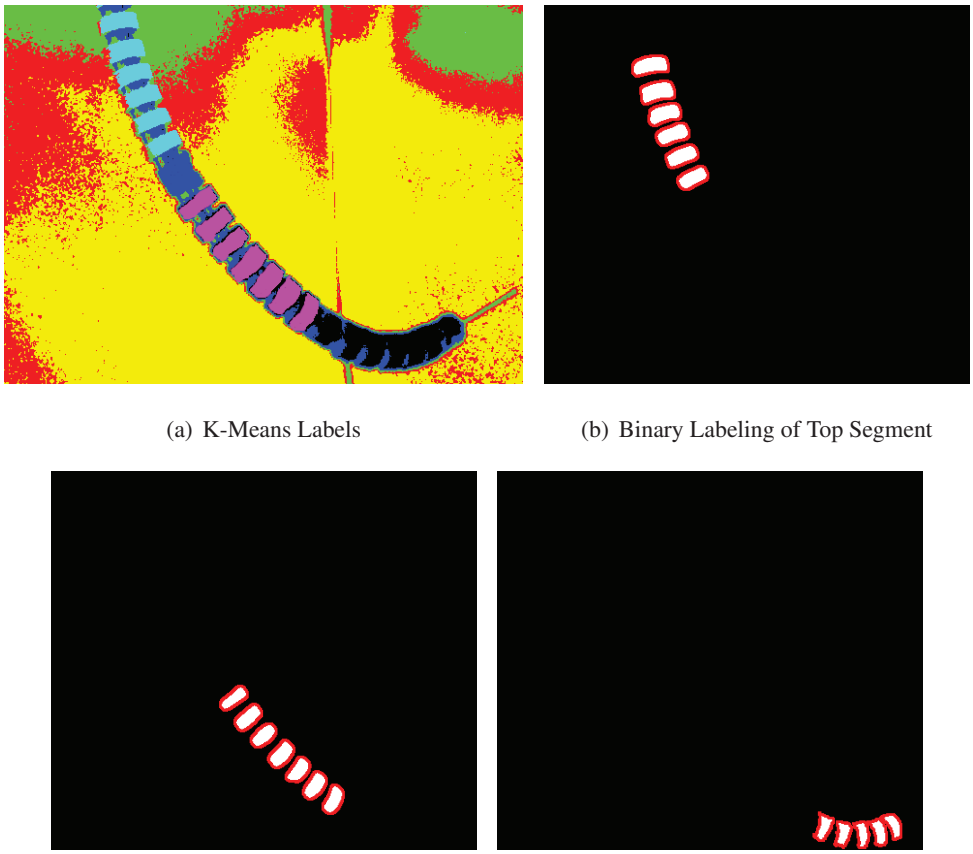
7.3 Parameterized Manifold

In Ch. 6, we described the following forward mapping:

$$\mathbf{G}(\psi_\mathbf{k}) = L_k \tag{7.10}$$

where $\psi_\mathbf{k}$ is as described in (7.1), L_k is a compressed feature descriptor (through applying Principal Components Analysis (PCA) to a set of training descriptors to reduce the dimensionality, Sec. 7.4.3), and $\mathbf{G}$ is the manifold mapping. We showed that we can use training samples to learn $\mathbf{G}$ and interpolate estimated features L_k based on arbitrary DOFs $\psi_\mathbf{k}$. We constructed a look-up table of these interpolated features and estimated unknown configurations by finding the nearest neighbors of an observed descriptor. We used a weighted-average of the configurations that produced the nearest feature matches to recover the configuration angles. Because we were operating with 2-DOFs, the look-up table was reasonably sized in terms of memory. However, as more DOFs are added, this becomes intractable quite quickly. In this work, we with to interpolate the feature descriptor manifold by parameterizing using the 3-DOF configuration angles $\mathbf{e}_k$

Ideally, we would like to learn the inverse mapping $\mathbf{G}^{-1}$ mapping features directly to the estimated rotation angles corresponding to that feature, thereby bypassing any need for a lookup table. Because these features are normally high-dimensional, the interpolation becomes quite complex. We found that using Radial Basis Functions (RBFs) [Buhmann, 2003] provided a good approximation to this interpolated mapping



(a) K-Means Labels (b) Binary Labeling of Top Segment

(c) Binary Labeling of Middle Segment (d) Binary Labeling of Bottom Segment

Figure 7.3: The binary labelings [**white pixels**] resulting from the clustering labeling shown in Fig. 7.3(a), and the associated edges of these binary masks [**red pixels**], obtained by applying the Sobel operator. The edges are used to build the feature descriptors, described in Sec. 7.4.2

if the underlying assumptions of smoothness are sufficiently maintained. As we will show, the descriptor choice becomes incredibly important as features which don't correspond smoothly to DOFs cannot be interpolated accurately.

7.4 Stereo Feature Descriptors

In order to build a feature descriptor which describes the physical configuration of our continuum segment, we first must select which pixels we will use to build the feature.

7.4.1 Image Segmentation

We use the same image segmentation procedure as in the single-segment case, described in detail in Sec. 6.2.2.2. A sample K-Means clustering result from our 3-segment robot is shown in Fig. 7.3(a), where different labels are automatically assigned to the pixels in the image according to their color separation. This facilitates segmenting the continuum segments from the image, individually, and results in 3 individual binary labeling, shown as white pixels in Figs. 7.3(b)-7.3(d). Next, as in Sec. 6.2.2.2, we again compute edges from these binary images using the Sobel edge detector. This gives us points along the outer-edges of the object, shown as red pixels in Figs. 7.3(b)-7.3(d).

7.4.2 Descriptor Extraction

The feature descriptor is the same concept as in the single-segment case, described in detail in Sec. 6.2.2.3. As before, our feature descriptor must be used to represent the 3D configuration of the snake segment by combining the monocular feature descriptors from the individual stereo cameras into a single, composite feature. This produces two separate views of the object in a stereo setup, and ambiguous movements due to out-of-plane perspective effects in one camera can be captured by the other camera by means of the descriptor.

A pictorial representation of this descriptor is shown in Fig. 7.4, whereby we show 14 bins for purposes of describing the idea, however we used 120-bins (every 3°) in our experiments. The bins radiate outwards from the center of the edge pixels (e.g., the *median* of the points). The bins count the number of edge pixels in each angular range. The histogram should be densely binned so that small changes in shape are captured and the descriptor is sensitive, yet not overly noisy.

Because this descriptor is sensitive to the location of the center point which we use to compute the radial bins, we perform a smoothing operation to provide some invariance to the location of the center that we use. We apply a Gaussian weighting around a bin location when a vote is added to the histogram, performing wrap-around at the boundaries of the histogram to account for the cyclic nature of rotations. We show results on the overall descriptor stability in Sec. 7.5.3.

7.4.3 Feature Pre-Processing

Before constructing the mapping $\mathbf{G}^{-1}$ of features to rotation angles, we pre-process the features to reduce the complexity using PCA. We use a magnetic sensor to provide ground truth pose angles for training purposes. To collect training data, we collect pairs of stereo images $S_i = \{I_{Li}, I_{Ri}\}$ and associated $\mathbf{e}_{k_i} = \{\alpha_{e_{k_i}}, \beta_{e_{k_i}}, \gamma_{e_{ki}}\}$ measurements, where $i = 1, \ldots, N$ for N discrete training samples.

The stereo pairs S_i are mapped to 1-D stereo feature descriptors (for each segment k) $\tilde{H}_{k_i} = [H_{k_{Li}}^T, H_{k_{Ri}}^T]$ to represent the shape of the segment on that frame. Our training set then becomes $\mathbf{A} = \{(\mathbf{e}_{k_1}, \tilde{H}_{k_1}), \ldots, (\mathbf{e}_{k_N}, \tilde{H}_{k_N})\}$. We use PCA to reduce each $\tilde{H}_{k_i}$ because each is very high-dimensional but also quite sparse. It is shown in [Murase and Nayar, 1995] that a compact representation of an object's appearance sufficiently captures the pose by creating a parametric eigenspace to represent this appearance. Therefore, we compute the principal components of the training set of $\tilde{H}_{k_i}$ samples. We observed that we can substantially reduce the dimensionality while still recovering a large percentage of the variance.

We then project the original training feature descriptors in $\mathbf{A}$ to the eigen-subspace:

$$L_{k_i} = E^T(\tilde{H}_{k_i} - c) \tag{7.11}$$

where c is the mean feature descriptor over all $\tilde{H}_{k_i}$, E is a matrix who's columns consist of the b eigenvectors corresponding to the top b eigenvalues we wish to preserve, and $L_{k_i} \in \mathbb{R}^{b \times 1}$. We choose b to capture a sufficient percentage of the variance of the original feature training set. We call these L_{k_i} samples *eigen-features* and these are used to learn the mappings in the parametric manifold. Next we describe the RBF

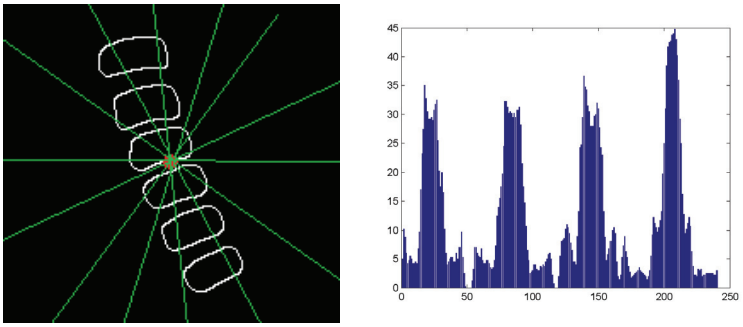


Figure 7.4: A pictorial representation of our feature descriptor, for a single segment of the robot from the left camera. Although we portray 14 bins, for purposes of describing the idea, in our experiments we used 120 bins.

method used to learn this mapping.

7.4.4 RBF Interpolation

A Radial Basis Function (RBF) is a function based on a scalar radius:

$$\phi(r) = \phi(|x - x_i|) \tag{7.12}$$

where $\phi(r)$ can be different types of functions, such as Gaussian, Multiquadric, Linear, Cubic, or Thinplate Splines. In order to use RBFs to interpolate a function mapping $f(x)$, an approximation of the function is made by choosing coefficients C_0, C_1, and λ_i to match values of the function at interpolation nodes, such as:

$$f(x) = C_0 + C_1 x + \sum_{i=1}^{n} \lambda i \phi(|x - x_i|) \tag{7.13}$$

Once coefficients C_0, C_1, and λ_i are found, we can use this to approximate the function at any point. As mentioned before, we seek the mapping

$$\mathbf{e}_k = \mathbf{G}^{-1}(L_k) \tag{7.14}$$

so that the estimate of the configuration of a particular segment can be directly computed from the associated stereo feature descriptor extracted from a pair of images. We use RBFs to interpolate this mapping $\mathbf{G}^{-1}$ from the training data described in section 7.4.3. This produces a multi-dimensional interpolation procedure mapping descriptor measurements in $\mathbb{R}^b$ to configurations in $\mathbb{R}^3$. Therefore we need to compute 3 different sets of RBF coefficients corresponding to each of the 3 rotation DOFs in $\mathbf{e}_k$. More specifically, one set of coefficients will interpolate the mapping of L_k to α_{e_k}:

$$\alpha_{e_k} = \tilde{\mathbf{G}}_{\alpha}^{-1}(L_k) \tag{7.15}$$

such that $\tilde{\mathbf{G}}_{\alpha}^{-1}$ is an approximation of the true mapping, $\mathbf{G}_{\alpha}^{-1}$, which estimates the angles α_{e_k} from L_k. We do the same for $\tilde{\mathbf{G}}_{\beta}^{-1}$ and $\tilde{\mathbf{G}}_{\gamma}^{-1}$, which approximate $\mathbf{G}_{\beta}^{-1}$ and $\mathbf{G}_{\gamma}^{-1}$.

In the end, we have coefficients which can efficiently produce estimates of rotations from any snake segment to the static base coordinate frame. This method is in place of our previous approach, which required a look-up table to be constructed from the forward mapping $\mathbf{G}$ and is sensitive to the density of the training samples. This means isolated training measurements may provide larger errors when interpolating

from the nearest-neighbors because of the weighted averaging of nearby samples. However, interpolations using RBFs perform very poorly with descriptors that don't vary smoothly (see Sec. 7.5.4).

7.5 Experiments

We experimented with the 3-segment continuum robot of Fig. 7.1, where each segment was coded with individual color markers to facilitate image segmentation. This type of marker-based extraction is common for surgery [Wei *et al.*, 1997]. In our experiments we used 2 PointGrey Dragonfly cameras for our stereo imaging. Four magnetic sensors were embedded at points known along the robot arm to produce ground truth of *relative* 3D rotations from each segment to its most proximal segment. We used the Ascension Technology 3D Guidance trakSTAR for ground truth angles, containing 4 Model 90 6-DOF sensors. Position static accuracy of the sensor is 1.4 mm RMS with an orientation static accuracy of 0.5°.

We call the most proximal segment, attached to the static robot base, segment 1. The next distal segment is called segment 2, and the final, distal-most segment is segment 3. Therefore, we seek rotations $\mathbf{R}_1^0$, $\mathbf{R}_2^0$, and $\mathbf{R}_3^0$, as described in (7.5). For ground truth, $\mathbf{R}_1^0$ is given directly by the magnetic trackers. To get $\mathbf{R}_2^0$, we are given $\mathbf{R}_2^1$ from the magnetic trackers and then compute

$$\mathbf{R}_2^0 = \mathbf{R}_1^0 \mathbf{R}_2^1 \tag{7.16}$$

and similarly for $\mathbf{R}_3^0$, we compute

$$\mathbf{R}_3^0 = \mathbf{R}_1^0 \mathbf{R}_2^1 \mathbf{R}_3^2 = \mathbf{R}_2^0 \mathbf{R}_3^2 \tag{7.17}$$

Then, using the Euler angle conversion expressed in (7.6), (7.7), and (7.8), we collect $\mathbf{e}_1^0$, $\mathbf{e}_2^0$, and $\mathbf{e}_3^0$ for use in training.

The color codings help extract the segments of the snake, individually from each other, using the image processing method described in Sec. 7.4.1. To begin, we train an RBF to interpolate the mapping $\mathbf{G}^{-1}$ by collecting descriptors from the stereo images, as described in Sec. 7.4.2, and then pre-processing them as in Sec. 7.4.3. For each frame, we get **three** stereo descriptors, each with their respective rotation angles to the static base frame from the magnetic trackers. Because the segments are treated independently, as a freely-rotating body in 3D space, for each stereo pair we get three completely separate training samples, corresponding to each of the 3 segments of the continuum arm. Because a descriptor is mapped to the

rotation of the end disk of that segment to the static base (regardless of the segments above it), we can collect these samples and train an RBF to interpolate the function mapping, as described in Sec 7.4.4. In this way, we remove the coupling effects that would be present with relative rotations. One advantage of treating the segments individually is that if one or more is occluded, we can still compute the configuration of the visible segments because the coupling of the proximal segments above it is accounted for in our formulation.

Our experiment consisted of randomly sampling the workspace of the continuum robot and collecting measurements for off-line training and testing. Overall, we collected 3483 total samples. We experimented with different amounts of training data to analyze the effect on the interpolation quality. In each of the cameras we extracted feature descriptors using 120 bins (every 3°), and so the composite stereo features were 240-dimensional, before applying PCA.

7.5.1 Training Set Size

Fig. 7.5 shows our analysis of the effect of the number of training samples to interpolate $\mathbf{G}^{-1}$ using an RBF with respect to overall median rotational error (in all 3 dimensions of rotation combined). As you can see, even training with only 10% of the data (amounting to 348 samples), we can achieve an overall median accuracy of 2.77°. Our best result comes with more training data, not surprisingly, at $\sim 1.43°$ of rotational accuracy.

To be sure these experiments are fair, for each trial we randomly permute the data samples and randomly select some for training and the rest for testing, according to the split required by the experiment. We

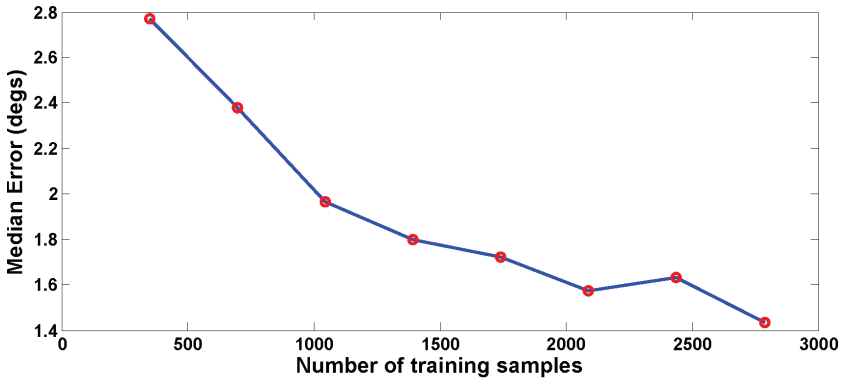


Figure 7.5: The effect of number of training samples on overall rotational accuracy.

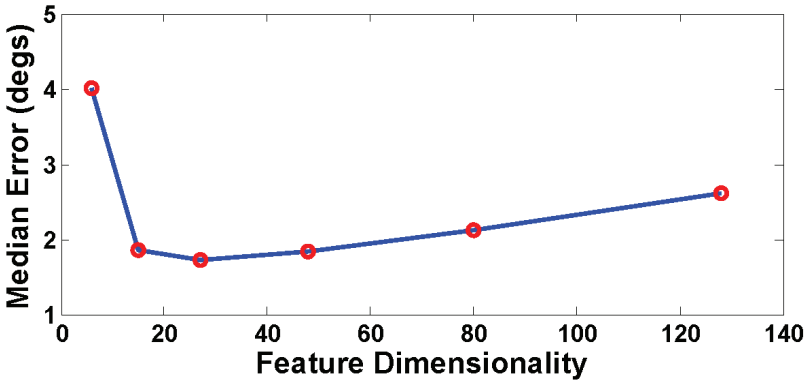


Figure 7.6: The effect of PCA feature dimensionality reduction on overall rotational accuracy.

then take the average result over this random sampling of the 3 trials to better capture the behavior of that experiment.

7.5.2 Dimensionality Reduction

We also experimented with the effect of the PCA dimensionality reduction on the 240-dimensional stereo feature descriptors. We chose to train with 50% of the data and test with the other 50% of the data, making sure the testing data was not included in the interpolation of the training data. Fig. 7.6 shows the effect of different levels of variance recovery on the overall median rotational accuracy. The recovered feature dimensionality increases as more variance is included in the PCA projections to produce the eigen-features L_k.

In these experiments, we notice an interesting result. As expected, using the same amount of training data in each experiment we get an increase in accuracy with more dimensions, but only up to a point, where at ~ 30 dimensions (corresponding to $80-85\%$ of the variance recovered) accuracy decreases. We attribute this effect to over-fitting. To investigate this further, we take the higher-dimensional cases and train with more data, and accuracy increases again to about the same level. This study gives a nice intuition about the trade-off between number of training samples and dimensionality reduction, and how they interact with each other. Again, as previously we randomly permute the training and testing samples.

7.5.3 Descriptor Stability

The descriptor histogram is built-up by the angles of the segmented pixel locations from the edge maps with respect to the center of the segment in the image. Typically we compute this center location by taking the *median* of the edge pixel locations, to approximate the center of the segment. As we noted earlier, the repeatability of this histogram to describe a particular configuration of a snake segment is sensitive to this center location. Therefore we added a smoothing operation to add some translational invariance to this center location.

To analyze the stability of the descriptor, we devised an experiment as follows. Given a configuration of a snake segment as a test, we compute the descriptor from a single camera as we already described. This is our reference histogram $H_{ref} \in \mathbb{R}^{120}$. We then add varying amounts of Gaussian noise to the median center location, using noise levels of 0.1 up to 8 pixels, and recompute the histogram with the noisy center H_n. Because the units of the descriptor are hard to interpret, we chose to use a percent difference to analyze how the descriptor changes with noisy center locations. The results are shown in Fig. 7.7. It is observed that the percent difference of the histograms computed with the noisy centers changes linearly with pixel noise. The slope of the best-fit line to this data is 5.2, meaning that for every pixel offset from the true center we get about an extra 5% difference in the histogram.

The percent difference (PD) is an average of the percent difference of every bin in H_n (for noise level n) to the reference histogram H_{ref}:

$$PD = \frac{\sum_{i=1}^{120} PD_i}{120} \tag{7.18}$$

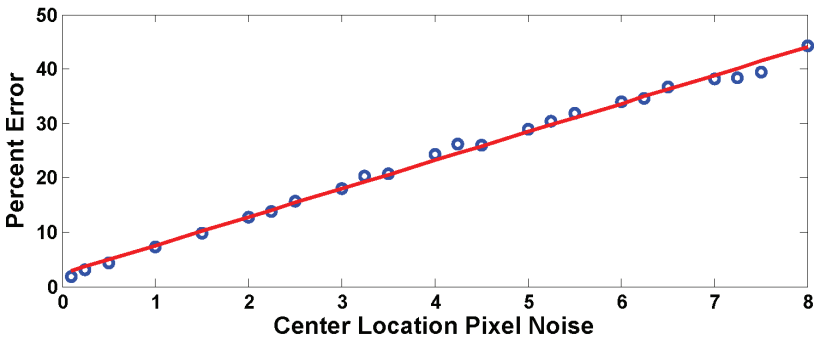


Figure 7.7: The stability of our feature descriptor with respect to the noise of the center pixel location.

where PD_i is the percent difference of a particular bin $i \in [1,\ldots,120]$ of the histograms:

$$PD_i = \frac{|H_n(i) - H_{ref}(i)|}{H_{ref}(i)} \times 100 \tag{7.19}$$

For each noise parameter, we perform 400 tests with that same noise parameter and take the average PD, again to reduce randomness. Using 0.5 pixels of noise, the PD from the reference is 4.3% and when we apply 2 pixels of noise, the PD grows to 12.8%. Although this still produces a larger percent difference in the descriptor than we would like, the smoothing operation reduces the tendency of the histograms to change significantly when the segment center is noisy. Further increasing this stability is a topic of future research.

7.5.4 Alternative Feature Descriptors

The choice of our descriptor is not arbitrary. To prove this, we chose a popular descriptor in the computer vision literature called *Histogram of Oriented Gradients* (HoG) [Dalal and Triggs, 2005]. In order for the interpolation mapping to work, the descriptors must vary smoothly with changes in 3D rotation.

We performed the same tests as previously described using HoG descriptors in place of ours. To contrast our approach, we update the **gradient** orientation bins using the gradient magnitude as the vote for that bin using the gray values of the image pixels. Again we combine the HoG's from the two cameras to construct a stereo feature descriptor, ensuring fair comparison. We found that the RBF cannot accurately interpolate the function mapping, and gives rotational errors on the order of $\sim 100^\circ$ and more. We believe this is the case because the HoG descriptor more accurately describes texture rather than shape, and this provides no smoothness guarantees in the parametric rotation space. This means that nearby HoG's in feature descriptor space do not necessarily describe similar physical configurations in terms of 3D rotation, but rather in appearance, which is not relevant to the problem being addressed in this work. It's important then that the feature descriptor is more of a *shape* descriptor rather than a *texture* descriptor.

7.6 Conclusions

In this chapter, we have described an algorithm to compute the configuration of a multi-segment continuum robot using vision. We constructed a function mapping which trains off of ground truth data to produce 3D rotation angles directly from a stereo feature descriptor. In the future, we plan to run closed-loop live experiments with a snake robot to perform fine-scale manipulations, as in [Bajo *et al.*, 2011], where the

algorithm described in this work can directly feed into the controller. To do this accurately over long periods of time, we will need to develop a tracking algorithm, noting that during a smooth motion of the continuum arm, the configuration angles will change slowly and the estimates may be filtered to avoid jumps and outliers.

Part III

Conclusions

Chapter 8

Conclusions

In this thesis, I have presented computer vision tools to enhance the surgical experience for a variety of different situations. Various tool tracking algorithms were described and shown to be successful in realistic in-vivo scenarios. Each tracker differs from each other depending on the requirements of the use-case and the procedure. No one algorithm is designed to solve all of tool tracking at once, but rather excel in a particular *type* of application. The algorithms complement each other in various ways, but all share the common theme of **learning**. I believe that any computer vision algorithm will be significantly limiting without the ability to learn. Whether it learns off-line or adapts on-the-fly, the skills to tune to particular objects or environments significantly increases the chances for success.

Whenever possible I showed stress-testing on the algorithm to assess the strengths and weaknesses and show areas for improvement as well as where the algorithm excels. For *LCT*, I was fortunate to be able to demonstrate the algorithm running live in a surgical procedure. This is the best way to analyze a medical vision approach, and I found the experience both educational and rewarding. I believe these tracking ideas could be implemented on a commercial system and disseminated on a large scale, like the da Vinci®. The tracking algorithms could be used as-is, in the current state-of-the-art approaches to surgery (e.g., both manual and robotic), as well as in future more automated systems.

The work presented with the IREP, and other similar continuum robot designs, is a necessary tool in being able to implement automated surgical tasks. I believe the field is going in this direction, because of the reduction in invasiveness, the increase in dexterity and safety, and the automated implementation of simple, routine tasks into the surgical plan. Once deemed safe enough, I believe robotic surgeons will welcome the opportunities afforded by, for example, automatically performing suturing and detecting and cleaning bleeds

”in the background” while the rest of the procedure continues forward. The hope is that this work helps to speed up operations, causes less distractions, results in less post-operative pains, and reduces costs due to shorter hospital stays and shorter operating times. I believe that using visual feedback is ideal because it doesn't require complicated and expensive hardware (e.g., magnetic trackers), which require constraints on sterilizability, and is more easily ported between different mechanical systems.

8.1 Limitations

Every algorithm has its limitations, and it's important to study and understand these in order to move forward in the field. Tool tracking in-vivo is a hard problem, which encounters may kinds of situations and challenges. Although the approaches presented in this thesis begin to address some of these issues, they aren't perfect and so it's worth discussing some of these limitations.

Feature Flow (Ch. 3) provides 2D information and learns the appearance of the tool on-the-fly. Although it requires minimal up-front information, ideally we would require no information at all. The reason is that, other than the initial inconvenience, it does happen that the tracker either loses track or gets ”stuck” on the wrong portion of the scene. In this situation, it is up to the user to notice and fix this by providing a new initialization to the tracker. Ideally, the system would both know with better confidence when it is stuck (which the tracker does attempt to accomplish) as well as find some way to re-initialize itself with its prior knowledge. In addition, it is a memory intensive approach because the database stores as much information as possible to maintain states of the track. One potential solution to this is to combine the ideas of machine learning and classification with the database to *compress* the information for each of the track states.

VTT (Ch. 4) provides 3D information in the form of joint angles for articulated robotic tools by using a kinematically-driven graphical renderer and virtual templates which can be matched to real image data. One of the the obvious limitations is the computational speed of processing each video frame, which precludes the algorithm from being run at video rates (e.g., ~ 30 frames/second). There are several potential improvements which may address this: first, a more intelligent search optimization routine could be used which ignores searching unrealistic configurations, and this may be driven by the kinematics estimate. Next, as the template matches are parallel in nature, a GPU may be utilized which could significantly speed up the per-frame processing time and push this forward towards real-time rates. Another potential improvement is to improve the realism of the graphical renderer so that the virtual templates match the image data even better than they

do in the currently-described implementation.

LCT (Ch. 5) also provides 3D information and was proven to be the most successful of the three tool tracking approaches, as it was demonstrated live during a porcine surgical procedure. However, there are still some limitations which could improve the performance even further. One is that the offline training procedure is quite arduous, and is specific to the features which are chosen. In this way, each new tool with novel features requires a new training of the classifier. It would be ideal if this training could be more automatic, combining some of the ideas of *VTT* by using the graphical renderer to train the features with the classifier and still remain detectable on real imagery. Although the processing time is slightly higher than we would like it to be, there are many parallelizable parts of the algorithm which could be ported to the GPU to speed this up. Finally, the tracker may take a little too long to acquire the tool at certain times. This is often due to the fact that the tool is either too far away from the camera, making the features too small to detect, or only views of the tool which were never learned are visible in the camera, and so the feature classifier has nothing to go on to initialize the tracker. By finding new ways to initialize the tracker, the overall performance and tracking robustness could be significantly improved.

Finally, the work presented on continuum robot configuration estimation shows both much improved accuracy over the state-of-the-art described in the literature as well as more realistic scenarios (e.g., out-of-plane motion, standard stereo cameras). However, one of the most significant limitations is the reliance on the camera viewpoint used during training. Because the stereo descriptors don't use any camera calibration data, the feature manifold is specific to that camera viewpoint, and so if/when the cameras move (which is inevitable), the feature manifold becomes invalid for these new viewpoints. One solution to this is to train *on-the-fly*, again using a graphical renderer by creating a local feature manifold from the current kinematic estimate on any given frame, and adjusting this as the camera moves. Additionally, although we show some robustness to occlusions, the feature descriptor is reliant on accurate localization of the continuum segment centroid, which moves (incorrectly) as more of the pixels are occluded. By improving the centroid localization method, we can improve the overall stability of the descriptors and subsequently, the resulting configuration accuracy of the snake robot arms.

8.2 Next Steps

I feel that the work presented in this thesis is a good step in the right direction towards improving both the tool tracking and visual sensing capabilities of continuum robots. Tool tracking must continue to use ideas in machine learning as well as different types of features simultaneously. One of the themes of the current literature is using one feature to achieve entire tasks (and simple features at that). When looking at real surgical imagery, it becomes immediately apparent that one feature does not suffice for any full computer vision task. I think the field will continue towards this multi-feature, visual learning direction, although it will likely vary between those who approach from the *bottom-up* (e.g., using low-level features to build up the pose) and those who believe in *top-down* methodologies (e.g., template matching, analyzing large-scale geometries of the tool).

As in most machine learning approaches, one of the major limitations is the availability of good training data in large quantities. When it comes to 2D training data, it's realistic to assume one may sit and click on lots of video sequences and collect ground truth. However, collecting 3D ground truth for articulated robots (e.g., joint values for the kinematic chain) is quite difficult. Typically, motion capture systems can provide this, but these are expensive and often difficult to use. On the other hand, with the availability of inexpensive depth cameras, like the Microsoft Kinect, I think we are starting to move closer to better ground truth. This should start to spur newer ideas with richer ground truth data for training better algorithms.

Finally, as newer applications and sensory devices arise, I believe different approaches to tracking and detection will be explored. For example, as 3D information becomes ever more prevalent in our lives, it's only a matter of time until we achieve this intra-operatively. When this day comes, and we are able to collect real-time, dense 3D information, the field will quickly begin to move in that direction. Tool tracking will most certainly be performed in 3D and become easier, and surgical interventions involving computer vision will become ubiquitous.

8.3 The Future of Robotic Surgery

Robotic Surgery has come a long way since the da Vinci® was introduced in 1999. Using a micro-processor to augment the abilities of the human surgeon was an obvious step towards improving the over surgical experience for both the physician and the patient, however many challenges, both mechanical and algorithmic, have affected the growth of the field. Neither hardware nor software alone will be able to solve this problem

entirely, and some of the best results have been shown with systems that display sophistication in both fields simultaneously. Currently, we are in a state of complete tele-operation whereby the surgeon is in control at all times. Although this comes with its advantages, further improvement will come with semi-automation, all towards the ultimate goal of fully-automated robotic surgery.

Semi-automation of simple tasks will allow the surgeon to concentrate on the more complicated, pressing needs of the operation. Many tasks are fairly repetitive and will eventually be performed entirely by a machine. However, as routine as these simple tasks may seem, they show some variation across implementations. For example, automatically suturing a wound requires knowledge of the pattern of the wound opening, the depth with which the needle can safely pierce the organ's surface, the amount of surrounding free space with which the manipulators can perform the actions, and various other detailed information needed for a successful suture. Most, if not all, of these factors are completely *visual* by nature. Only with accurate computer vision algorithms can the robot safely determine these parameters before the path planning of the robot can even be considered. Similarly for blood suctioning, first the robot must detect and localize the bleed before the robot can perform the actual suctioning. And so in order for us to achieve a reasonable level of automation for even the simplest surgical tasks, we must first refine and mature the visual intelligence of the surgical robots, an effort towards which I hope this thesis contributes at least some significant role.

Given the current state-of-the-art of the robotics and vision communities, I think that it's possible we will see a robot autonomously perform routine operations, but I don't believe we will see a fully-functional automated surgical robot in our lifetime that can react as intelligently as a skilled human surgeon. At this point in time, I don't think there's a single person who is willing to relinquish complete control to a machine in their most vulnerable of moments. When a patient receives a surgical intervention as a treatment, priorities change, health-care costs become irrelevant, and we seek the best medical care money can buy. Before we even consider going down this road, I propose we implement a *Turing Test for Surgery*: if a surgical expert cannot distinguish between a robot-performed and human-performed surgical operation based on the outcome alone, we have achieved our goal of sufficiently intelligent surgical robots. Until then, please keep the human at the helm.

Part IV

Bibliography

Bibliography

[Agrawal *et al.*, 2010] V. Agrawal, W. J. Peine, B. Yao, and S. Choi. Control of cable actuated devices using smooth backlash inverse. In *IEEE International Conference on Robotics and Automation*, pages 1074–1079, 2010.

[Bajo *et al.*, 2011] A. Bajo, R. E. Goldman, and N. Simaan. Joint and configuration feedback for enhanced performance of multi-segment continuum robots. In *IEEE International Conference on Robotics and Automation*, 2011.

[Barron *et al.*, 1994] J. L. Barron, S. S. Beauchemin, and D. J. Fleet. On optical flow. In *International Conference on Artificial Intelligence and Information-Control Systems of Robots*, pages 3–14, 1994.

[Belongie *et al.*, 2002] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, April 2002.

[Bosch *et al.*, 2007] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *ACM International Conference on Image and Video Retrieval*, 2007.

[Bresenham, 1965] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, January 1965.

[Buhmann, 2003] M. D. Buhmann. *Radial Basis Functions*. Cambridge University Press, 2003.

[Burschka *et al.*, 2004] D. Burschka, J. J. Corso, M. Dewan, W. Lau, M. Li, H. Lin, P. Marayong, N. Ramey, G. D. Hager, B. Hoffman, D. Larkin, and C. Hasser. Navigating inner space: 3-d assistance for minimally invasive surgery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[Camarillo *et al.*, 2008a] D. B. Camarillo, K. E. Loewke, C. R. Carlson, and J. K. Salisbury. Vision based 3-d shape sensing of flexible manipulators. *IEEE International Conference on Robotics and Automation*, pages 2940–2947, May 2008.

[Camarillo *et al.*, 2008b] D. B. Camarillo, C. F. Milne, C. R. Carlson, M. R. Zinn, and J. K. Salisbury. Mechanics Modeling of Tendon-Driven Continuum Manipulators. *IEEE Transactions on Robotics*, 24(6):1262–1273, 2008.

[Collins *et al.*, 2005] R. Collins, Y. Liu, and M. Leordeanu. Online selection of discriminative tracking features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1631–1643, 2005.

[Cortes and Vapnik, 1995] C. Cortes and V. N. Vapnik. Support-vector networks. *Machine Learning*, 20, 1995.

[Croom *et al.*, 2010] J. M. Croom, D. C. Rucker, J. M. Romano, and R. J. Webster III. Visual sensing of continuum robot shape using self-organizing maps. In *IEEE International Conference on Robotics and Automation*, pages 4591–4596, 2010.

[Dalal and Triggs, 2005] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[Doignon *et al.*, 2004] C. Doignon, F. Nageotte, and M. de Mathelin. Detection of grey regions in color images: application to the segmentation of a surgical instrument in robotized laparoscopy. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[Doignon *et al.*, 2005] C. Doignon, P. Graebling, and M. de Mathelin. Real-time segmentation of surgical instruments inside the abdominal cavity using a joint hue saturation color feature. *Real-Time Imaging*, 11(5-6):429–442, 2005.

[Doignon *et al.*, 2006] C. Doignon, F. Nageotte, and M. de Mathelin. Segmentation and guidance of multiple rigid objects for intra-operative endoscopic vision. In *European Conference on Computer Vision*, pages 314–327, 2006.

[Duda *et al.*, 2001] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2 edition, 2001.

[Dupont *et al.*, 2010] P. Dupont, J. Lock, B. Itkowitz, and E. Butler. Design and Control of Concentric-Tube Robots. *IEEE Transactions on Robotics*, 26(2):209–225, 2010.

[Fischler and Bolles, 1981] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[Gravagne and Walker, 2002] I. A. Gravagne and I. D. Walker. Manipulability, force, and compliance analysis for planar continuum manipulators. *IEEE Transactions on Robotics and Automation*, 18(3), June 2002.

[Groeger *et al.*, 2008] M. Groeger, K. Arbter, and G. Hirzinger. Motion tracking for minimally invasive robotic surgery. *Medical Robotics, I-Tech Education and Publishing*, pages 117–148, 2008.

[Hannan and Walker, 2003] M. Hannan and I. Walker. Vision based shape estimation for continuum robots. In *IEEE International Conference on Robotics and Automation*, pages 3449–3454, 2003.

[Hartley and Zisserman, 2003] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[Hatzinger *et al.*, 2006] M. Hatzinger, S. T. Kwon, S. Langbein, S. Kamp, A. Hacker, and P. Alken. Hans christian jacobaeus: Inventor of human laparoscopy and thoracoscopy. *Journal of Endourology*, 20(11):848–850, November 2006.

[Hinterstoisser *et al.*, 2011] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *IEEE International Conference on Computer Vision*, 2011.

[Hirose, 1993] S. Hirose. *Biologically Inspired Robots: Snake-like Locomotors and Manipulators*. Oxford University Press, USA, 1993.

[Horn and Schunck, 1981] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[Int,] Intuitive Surgical, Inc. `http://www.intuitivesurgical.com/`.

[Jones and Walker, 2006] B. A. Jones and I. D. Walker. Practical kinematics for real-time implementation of continuum robots. *IEEE Transactions on Robotics*, 22(6):1087–1099, December 2006.

[Kesner and Howe, 2010] S. B. Kesner and R. D. Howe. Design and control of motion compensation cardiac catheters. In *IEEE International Conference on Robotics and Automation*, pages 1059–1065, 2010.

[Kesner and Howe, 2011] S. B. Kesner and R. D. Howe. Position control of motion compensation cardiac catheters. *IEEE Transactions on Robotics*, 27(6):1045–1055, 2011.

[Krupa *et al.*, 2003] A. Krupa, J. Gangloff, C. Doignon, M. de Mathelin, G. Morel, G. Morel, J. Leroy, and L. Soler. Autonomous 3-d positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. *IEEE Transactions on Robotics and Automation, Special Issue on Medical Robotics*, 19(5):842–853, 2003.

[Lee *et al.*, 1994] C. Lee, Y. F. Wang, D. Uecker, and Y. Wang. Image analysis for automated tracking in robot-assisted endoscopic surgery. In *International Conference on Pattern Recognition*, 1994.

[Lehman *et al.*, 2008] A. C. Lehman, N. A. Wood, J. Dumpert, D. Oleynikov, and S. M. Farritor. Robotic natural orifice translumenal endoscopic surgery. In *IEEE International Conference on Robotics and Automation*, 2008.

[Lepetit and Fua, 2006] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, 2006.

[Lowe, 2004] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[Lucas and Kanade, 1981] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, volume 81, pages 674–679, 1981.

[Mack, 2001] M. J. Mack. Minimally invasive and robotic surgery. *The Journal of the American Medical Association*, 285:568–572, 2001.

[Marr and Hildreth, 1980] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London*, 207(1167):187–217, 1980.

[Murase and Nayar, 1995] H. Murase and S. K. Nayar. Visual learning and recognition of 3d objects from appearance. *International Journal on Computer Vision*, 14(1):5–24, January 1995.

[Nageotte *et al.*, 2005] F. Nageotte, C. Doignon, M. de Mathelin, P Zanne, and L Soler. Circular needle and needle-holder localization for computer-aided suturing in laparoscopic surgery. In *Proceedings of the SPIE International Symposium on Medical Imaging*, 2005.

[Nageotte *et al.*, 2009] F. Nageotte, L. Ott, P. Zanne, and M. de Mathelin. Analysis and improvement of image-based insertion point estimation for robot-assisted minimally invasive surgery. In *IEEE International Conference on Robotics and Automation*, 2009.

[Ning *et al.*, 2008] H. Ning, W. Xu, Y. Gong, and T. S. Huang. Discriminative learning of visual words for 3d human pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[Ohbuchi *et al.*, 2008] R. Ohbuchi, K. Osada, T. Furuya, and T. Banno. Salient local visual features for shape-based 3d model retrieval. In *IEEE International Conference on Shape Modeling and Applications*, 2008.

[Pennec *et al.*, 2006] X. Pennec, P. Fillard, and N. Ayache. A riemannian framework for tensor computing. *International Journal of Computer Vision*, 66(1):41–66, 2006.

[Pezzementi *et al.*, 2009] Z. Pezzementi, S. Voros, and G. Hager. Articulated object tracking by rendering consistent appearance parts. In *IEEE International Conference on Robotics and Automation*, 2009.

[Poi,] Point Grey Research Inc. `http://www.ptgrey.com//`.

[Reiter and Allen, 2010] A. Reiter and P. K. Allen. An online approach to in-vivo tracking using synergistic features. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[Reiter *et al.*, 2011] A. Reiter, R. E. Goldman, A. Bajo, K. Iliopoulos, N. Simaan, and P. K. Allen. A learning algorithm for visual pose estimation of continuum robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[Reiter *et al.*, 2012a] A. Reiter, P. K. Allen, and T. Zhao. Feature classification for tracking articulated surgical tools. In *International Conference on Medical Image Computing and Computer Assisted Intervention*, 2012.

[Reiter *et al.*, 2012b] A. Reiter, P. K. Allen, and T. Zhao. Learning features on robotic surgical tools. In *Workshop on Medical Computer Vision, IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[Reiter *et al.*, 2012c] A. Reiter, P. K. Allen, and T. Zhao. Marker-less articulated surgical tool detection. In *Computer Assisted Radiology and Surgery*, 2012.

[Reiter *et al.*, 2012d] A. Reiter, A. Bajo, K. Iliopoulos, N. Simaan, and P. K. Allen. Learning-based configuration estimation of a multi-segment continuum robot. In *IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2012.

[Romanelli and Earle, 2009] J. Romanelli and D. Earle. Single-port laparoscopic surgery: An overview. *Surgical Endoscopy*, 23(7):1419–1427, 2009.

[Rosten and Drummond, 2005] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.

[Shakhnarovich *et al.*, 2003] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *IEEE International Conference on Computer Vision*, 2003.

[Shotton *et al.*, 2011] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[Simaan *et al.*, 2004] N. Simaan, R. H. Taylor, and P. Flint. A dexterous system for laryngeal surgery. In *IEEE International Conference on Robotics and Automation*, pages 351–357, 2004.

[Simaan *et al.*, 2009] N. Simaan, K. Xu, A. Kapoor, W. Wei, P. Kazanzides, P. Flint, and R. Taylor. Design and integration of a telerobotic system for minimally invasive surgery of the throat. *The International Journal of Robotics Research*, 28(9):1134–1153, September 2009.

[Simaan *et al.*, 2012] N. Simaan, A. Bajo, A. Reiter, P. K. Allen, and D. Fowler. Lessons learned using the insertable robotic effector platform (irep) for single port access surgery. In *The Hamlyn Symposium on Medical Robotics*, 2012.

[Torr and Murray, 1997] P. H. S. Torr and D. W. Murray. The development and comparison of robust methods for estimating the fundamental matrix. *International Journal of Computer Vision*, 24(3):271–300, 1997.

[Tuzel *et al.*, 2006] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *European Conference on Computer Vision*, 2006.

[Tuzel *et al.*, 2007] O. Tuzel, F. Porikli, and P. Meer. Human detection via classification on riemannian manifolds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[Viola and Jones, 2004] P. A. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[Voros *et al.*, 2007] S. Voros, J. Long, and P. Cinquin. Automatic detection of instruments in laparoscopic images: A first step towards high-level command of robotic endoscopic holders. *The International Journal of Robotics Research*, 26(11-12):1173–1190, November 2007.

[Walker *et al.*, 2006] I. D. Walker, C. Carreras, R. McDonnell, and G. Grimes. Extension versus bending for continuum robots. *International Journal of Advanced Robotic Systems*, 3(2):171–178, 2006.

[Webster III and Jones, 2010] R. J. Webster III and B. A. Jones. Design and kinematic modeling of constant curvature continuum robots: A review. *The International Journal of Robotics Research*, June 2010.

[Webster III *et al.*, 2009] R. J. Webster III, J. M. Romano, and N. J. Cowan. Mechanics of Precurved-Tube Continuum Robots. *IEEE Transactions on Robotics*, 25(1):67–78, 2009.

[Wei *et al.*, 1997] G. Q. Wei, K. Arbter, and G. Hirzinger. Automatic tracking of laparoscopic instruments by color coding. In *Joint Conference on Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery*, pages 357–366, 1997.

[Wolf *et al.*, 2011] R. Wolf, J. Duchateau, P. Cinquin, and S. Voros. 3d tracking of laparoscopic instruments using statistical and geometric modeling. In *International Conference on Medical Image Computing and Computer Assisted Intervention*, 2011.

[Xu and Simaan, 2006] K. Xu and N. Simaan. Actuation compensation for flexible surgical snake-like robots with redundant remote actuation. In *IEEE International Conference on Robotics and Automation*, pages 4148–4154, May 2006.

[Xu *et al.*, 2009] K. Xu, R. E. Goldman, D. Jienan, P. K. Allen, D. L. Fowler, and N. Simaan. System design of an insertable robotic effector platform for single port access (spa) surgery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5546–5552, 2009.

[Yin and Collins, 2007] Z. Yin and R. Collins. On-the-fly object modeling while tracking. In *IEEE Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[Zhang and Payandeh, 2002] X. Zhang and S. Payandeh. Application of visual tracking for robot-assisted laparoscopic surgery. *Journal of Robotic Systems*, 19(7):315–328, 2002.

[Zhang, 2000] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[Zhao *et al.*, 2009a] T. Zhao, W. Zhao, D.J. Halabe, B.D. Hoffman, and W.C. Nowlin. Fiducial marker design and detection for locating surgical instrument in images. *US Patent*, US 2010/0168562 A1, 04 2009.

[Zhao *et al.*, 2009b] T. Zhao, W. Zhao, B. D. Hoffman, W. C. Nowlin, and H. Hui. Efficient vision and kinematic data fusion for robotic surgical instruments and other applications. *US Patent*, US 2010/0331855 A1, 06 2009.

[Zhao *et al.*, 2009c] T. Zhao, W. Zhao, and W.C. Nowlin. Configuration marker design and detection for instrument tracking. *US Patent*, US 2010/0168763 A1, 04 2009.