

**A Fast, Low-Cost, Computer Vision-Based Approach For
Tracking Surgical Tools**

**A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY**

Rodney Lee Dockter II

**IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE**

Timothy Kowalewski Ph.D.

August, 2013

© Rodney Lee Dockter II 2013
ALL RIGHTS RESERVED

Acknowledgements

Several groups and people deserve acknowledgment for their part in this work. The University of Minnesota Medical Devices and Robotics Lab, the CREST and SimPortal groups at the University of Minnesota Medical School, Dr. Robert Sweet, Mr. Troy Reihsen, Dr. Sanket Chauhan, the Medical Devices Center and Darrin Beekman, Will Durfee, Ph.D. Nikos Papanikolopoulos, Ph.D. and the University of Minnesota Mechanical Engineering and Computer Science departments.

Dedication

To my friends and family without whom this would not have been possible.

Abstract

The number of Robot-Assisted Minimally Invasive Surgery (RMIS) procedures has grown immensely in recent years. Like Minimally Invasive Surgeries (MIS), RMIS procedures provide improved patient recovery time and reduced trauma due to smaller incisions relative to traditional open procedures. Given the rise in RMIS procedures, several organizations and companies have made efforts to develop training tasks and certification criteria for the da Vinci robot. Each training task is evaluated with various quantitative criteria such as completion time, total tool path length and economy of motion, which is a measurement of deviation from an ‘ideal’ path. All of these metrics can benefit greatly from an accurate, inexpensive and modular tool tracking system that requires no modification to the existing robot.

While the da Vinci uses joint kinematics to calculate the tool tip position and movement internally, this data is not openly available to users. Even if this data was open to researchers, the accuracy of kinematic calculations of end effector position suffers from compliance in the joints and links of the robot as well as finite uncertainties in the sensors. In order to find an accurate, available and low-cost alternative to tool tip localization, we have developed a computer vision based design for surgical tool tracking. Vision systems have the added benefit of being relatively low cost with typical high resolution webcams costing around 50 dollars. We employ a joint geometric constraint—probabilistic Hough transform method for locating the tool shaft and subsequently the tool tip.

The tool tracking algorithm presented was evaluated on both an experimental webcam setup as well as a da Vinci endoscope used in real surgeries. This system can accurately locate the tip of a robotic surgical tool in real time with no augmentation of the tool. The proposed algorithm was evaluated in terms of speed and accuracy. This method achieves an average 3D positional tracking accuracy of 3.05 mm and at 25.86 frames per second for the experimental webcam setup. For the da Vinci endoscope setup, this solution achieves a frame rate of 26.99 FPS with an average tracking accuracy of 8.68 mm in 3D and 1.88 mm in 2D. The system demonstrated successful tracking of RMIS tools from captured video of a real patient case.

Contents

Acknowledgements	i
Dedication	ii
Abstract	iii
List of Tables	viii
List of Figures	ix
1 Introduction	1
2 Literature Review	3
2.1 Overview	3
2.2 Training Modules	4
2.2.1 Mechanical Systems	15
2.2.2 Incorporated Simulator Systems	16
2.2.3 Ultrasound Systems	19
2.2.4 Visual Systems	20
2.2.5 Electromagnetic Systems	27
2.2.6 Comparison of Tool Tracking Technologies and Gap Analysis . .	29
2.3 Anatomy Tracking	31
2.4 Gap Analysis and Research Direction	37
2.4.1 Tool Tracking	37
2.4.2 Anatomy Tracking	39

2.4.3	Research Direction	40
3	Algorithm Design and Selection	42
3.1	Design Goals	42
3.2	Algorithm Development	44
3.3	2D Object Detection Step	46
3.3.1	Haar Feature Library Approach	46
3.3.2	Depth Segmentation Approach	49
3.3.3	Hough Transform and Geometric Approach	54
3.4	Algorithm Evaluation and Selection for Object Detection	60
3.5	Depth Extraction Step	67
3.6	Cartesian Registration Step	68
3.7	Hardware Design and Implementation	69
4	Implementation of the Complete Tracking Algorithm	75
4.1	Complete Algorithm Overview: Joint Hough Transform-Geometric Approach	76
4.1.1	Modified Depth Extraction Method and Calibration	83
4.1.2	Cartesian Registration Calibration	90
4.2	Software Design and Implementation	96
4.2.1	Main Entry Point Thread	99
4.2.2	Video Capture Thread	100
4.2.3	Right and Left Tracking Threads	102
4.2.4	3D Localization Thread	104
4.2.5	Conclusion	106
5	Experimental Verification and Performance Characterization	108
5.1	Performance Benchmarking Overview	108
5.2	Experimental Setup: Computational Speed	108
5.2.1	Computational Speed Analysis	111
5.3	Experimental Setup: Tracking Accuracy	114
5.3.1	Tracking Accuracy Analysis	119
5.4	Tracking Robustness Analysis	125

5.4.1	Performance Analysis	132
6	Implementation in da Vinci Surgical System	133
6.1	Surgical Endoscope Integration and Implementation	134
6.2	Surgical Endoscope Calibration	136
6.3	Surgical Endoscope Performance Assessment Setup	149
6.4	Surgical Endoscope Performance Analysis	151
6.4.1	Surgical Endoscope Computational Speed Performance	151
6.4.2	Surgical Endoscope Tracking Noise	155
6.4.3	Surgical Endoscope Tracking Accuracy	156
6.5	Tracking Performance Using Real Operating Room Footage	164
6.6	Surgical Endoscope Performance Conclusion	167
7	Conclusion and Discussion	170
7.1	Overview Of The Presented Methodology	170
7.2	Performance Analysis	172
7.3	Performance Conclusion	174
7.3.1	Future Work	175
	References	177
	Appendix A. Software Details	185
A.1	Software Implementation Details	185
A.1.1	Thread Creation	186
A.1.2	Custom Data Structures	187
A.1.3	Tracking Thread Functions	188
A.1.4	3D Localization Thread Functions	189
	Appendix B. Open Medical Interface	191
B.1	Initial Design Specification	191
	Appendix C. SurgTRAK Development	194
C.1	Overview	194

Appendix D. Glossary and Acronyms	196
D.1 Glossary	196
D.2 Acronyms	198

List of Tables

2.1	Tool Tracking Systems	31
2.2	Tracking Technology Gap	39
5.1	Computational Time Performance	114
5.2	Tracking Deviation	121
5.3	Tracking Accuracy	124
6.1	Computational Time Performance Endoscope	154
6.2	Tracking Deviation Endoscope	156
6.3	Overall Tracking Accuracy Endoscope	160
6.4	X Component Tracking Accuracy Endoscope	161
6.5	Y Component Tracking Accuracy Endoscope	162
6.6	Z Component Tracking Accuracy Endoscope	163
7.1	Tracking Performance Comparison	173
D.1	Acronyms	198

List of Figures

2.1	FLS Box	5
2.2	Simulab partial task	6
2.3	VRTrainer Setup	7
2.4	Mimic MdVT	8
2.5	FRS Module	10
2.6	Metric Equations	11
2.7	Path Length Graph	12
2.8	Tool Wrist	14
2.9	RedDragon Mechanism	16
2.10	Xitact ITP Mechanism	17
2.11	ProMIS tracking	18
2.12	CMS-HS Tracking System	19
2.13	Polaris Models	21
2.14	Polaris Vicra	22
2.15	Color Marker Tracking	23
2.16	Color Segmentation Tracking	24
2.17	Geometric Constraint Tracking	24
2.18	Model Based Feature Tracking	26
2.19	Polhemus Fastrak	28
2.20	Fiber Optic Probe	33
2.21	Ortmaier Visual Tracking	34
2.22	Monnich Visual Servoing	35
2.23	Light Striping Example	36
2.24	2 Source Periodic Example	37

3.1	Design Goals	43
3.2	da Vinci Tool Tip	44
3.3	Pixel Localization	45
3.4	Microsoft Lifecam	45
3.5	General Tracking Approach	46
3.6	Haar Training Method	47
3.7	Haar Training Image	48
3.8	Anatomical Background Image	48
3.9	Stereo Camera Setup	49
3.10	Disparity Diagram	50
3.11	Camera Coordinate System	51
3.12	Pixel Coordinate System	51
3.13	OpenCV Chessboard	53
3.14	Found Corners	53
3.15	Geometric Constraints	54
3.16	Tool Edges	55
3.17	Fast Gradient	56
3.18	Slow Gradient	56
3.19	Canny and Sobel	58
3.20	Hough Lines Example	59
3.21	Initial Haar Tracking Attempt	60
3.22	Adjusted Haar Tracking Attempt	61
3.23	Second Haar Tracking Attempt	61
3.24	Primary Intensity Regions	62
3.25	Depth Extraction	63
3.26	Depth Threshold	63
3.27	Depth Threshold 2	64
3.28	Initial Hough Test	65
3.29	Second Hough Test	65
3.30	Third Hough Test	66
3.31	Fourth Hough Test	66
3.32	Single Tool Disparity	67

3.33	Pixel Space With Global Space	69
3.34	Webcam Housing	71
3.35	Camera Mount Top	71
3.36	Camera Mount Base	72
3.37	Stereo Camera Mount	72
3.38	DVI Frame Grabber	73
3.39	da Vinci Endoscope	74
4.1	Elected Tracking Approach	75
4.2	Geometric Priors	77
4.3	End Points of a Hough Transform	78
4.4	Detection Steps	79
4.5	Tool Tip Detection	82
4.6	Modified Depth Extraction	84
4.7	New Calibration Board	84
4.8	Calibration Board Orientation	85
4.9	Calibration Board Rotation	86
4.10	Calibration Board Indices	86
4.11	Webcam Calibration Mount	87
4.12	Disparity and Depth	88
4.13	Disparity Depth Line Fit	89
4.14	X Location Plot	91
4.15	Y Location Plot	91
4.16	World Position Reprojected 141 mm	93
4.17	World Position Reprojected 181 mm	93
4.18	World Position Reprojected 222 mm	94
4.19	World Position Reprojected 259 mm	95
4.20	World Position Reprojected 299 mm	95
4.21	Alternate 299 mm Reprojection	96
4.22	Multi-Threaded Data Flow	97
4.23	OpenGL Real Time Graphing	98
4.24	Qt based GUI	98
4.25	Thread Creation	99

4.26	Data Structure Flow	100
4.27	Capture Thread Sequence	101
4.28	Buffer Sequence	102
4.29	Tracking Thread Cycle	103
4.30	Localization Thread Cycle	104
4.31	Tool Enumeration	105
4.32	Software Diagram	106
5.1	Timing Diagram	109
5.2	Tracking Thread Timing	110
5.3	Total Computation Time	112
5.4	FPS Performance	113
5.5	Experimental Trajectory Setup	115
5.6	Experimental Trajectory Board	116
5.7	Orientation of the Trajectory Board	116
5.8	Trajectory Board With Tool	117
5.9	Trajectory Path Models	117
5.10	Data vs. Model	118
5.11	Stationary Surgical Tool	118
5.12	Stationary Deviation at Various Depths	119
5.13	Small Trajectory at 205 mm	122
5.14	Middle Trajectory at 329 mm	122
5.15	Large Trajectory at 310 mm	122
5.16	Small Trajectory at 232 mm	123
5.17	Average Reprojection Errors	124
5.18	Percentage of Reprojection Errors Within 4 mm	125
5.19	Tool Configuration Test 1	126
5.20	Tool Configuration Test 2	126
5.21	Tool Configuration Test 3	127
5.22	Tool Configuration Test 4	127
5.23	Tool Configuration Test 5	127
5.24	Tool Configuration Test 6	128
5.25	Tool Configuration Test 7	128

5.26	Tool Configuration Test 8	129
5.27	Tool Configuration Test 9	129
5.28	Tool Configuration Test 10	130
5.29	Tool Configuration Test 11	130
5.30	Tip Detection Error	131
6.1	OR Implementation	134
6.2	Or Setting Video Capture	135
6.3	Revised Calibration Board	137
6.4	The Data Collection Setup	137
6.5	Lathe Diagram	138
6.6	Endoscope Calibration Setup	138
6.7	Endoscope Calibration Images	139
6.8	Endoscope Lens Divergence	139
6.9	Endoscope Depth and Disparity	140
6.10	Disparity Variations Over Pixel Space	141
6.11	Disparity Offsets Over Pixel Space	141
6.12	Polynomial Fit to Disparity Offset	142
6.13	Depth vs. Disparity Offset	144
6.14	X Pixel * World Z vs. World X	145
6.15	Y Pixel * World Z vs. World Y	146
6.16	Calibration Reprojection 150 mm	146
6.17	Calibration Reprojection 150 mm Alternate	147
6.18	Calibration Reprojection 225 mm	147
6.19	Calibration Reprojection 225 mm Alternate	148
6.20	Timing Diagram	149
6.21	Stationary Tool Noise Setup	150
6.22	Endoscope Tracking Accuracy Experiment	151
6.23	Total Computation Time Endoscope Setup	152
6.24	FPS Performance Endoscope	153
6.25	Stationary Deviation at Various Depths (Endoscopic)	155
6.26	Small Trajectory at 198 mm	157
6.27	Small Trajectory at 212 mm	157

6.28	Small Trajectory at 225 mm	158
6.29	Small Trajectory at 248 mm	158
6.30	Average Reprojection Errors	163
6.31	Sample Surgical Procedure Video Frame	165
6.32	Sample Tool Tracking <i>in vivo</i>	166
6.33	Lost Tool Tracking <i>in vivo</i>	166
7.1	Tool Tracking Review	171
7.2	Tool Tracking Method Overview	171
B.1	Design Overview	192
B.2	Data Packet Overview	193
C.1	SurgTRAK Diagram	195

Chapter 1

Introduction

Within the past two decades, advances in Minimally Invasive Surgery (MIS) and Robot-Assisted Minimally Invasive Surgery (RMIS) have greatly improved the success rate and patient experience in surgical procedures. Since these new surgical technologies cannot be taught with traditional Halstedian models, wherein an apprentice learns by watching a master, the demand for new training methods has also grown in order to teach surgeons safely and effectively. In order to train surgeons with a certain level of proficiency, various routines have been developed. These routines test various skills required of a surgeon using minimally invasive tools, such as accuracy and fluidity of motion. Accurate spatial and temporal motion data for tools and anatomy are crucial prerequisites to create quantitative metrics of performance and skill level of a surgeon. Extensive research and development has gone into the tracking of laparoscopic tools, but relatively little has gone into applicability and transferability to robotic systems. Even less developed is the task of visually tracking patient anatomy during a procedure. This work will present the development of an inexpensive, fast, and accurate computer vision system for tracking surgical robotic tools.

- Chapter 2 presents a review of the literature surrounding training systems for MIS and RMIS surgical skills, metrics use for training and prior work for tracking both surgical tools and anatomy.
- Chapter 3 provides an overview of the algorithm development, a brief analysis of the various algorithms, and hardware implementation for the tracking of surgical

tools.

- Chapter 4 provides an overview of the implementation of the tracking algorithm and the software design.
- Chapter 5 details the performance characteristics of the tracking algorithm. This includes experimental setups, accuracy, timing, footage and other results.
- Chapter 6 covers the use and implementation of the tracking system in an operating room setting with a da Vinci robot.
- Chapter 7 presents an analysis of the tracking system and suggestions for future work.
- Appendix A provides in-depth details of the software implementation of the tracking algorithm.
- Appendix B details work done concerning the development of an Open Medical Interface for developing cross platform, open-source medical training devices.
- Appendix C details additions made to the SurgTRAK data acquisition interface as a means for widespread distribution of the tool tracking algorithm.

Chapter 2

Literature Review

2.1 Overview

Just as the use of Minimally Invasive Surgery (MIS) techniques exploded in the 1990's, Robot-Assisted Minimally Invasive Surgery (RMIS) technologies have grown exponentially in recent years. The number of surgeries performed worldwide with the da Vinci surgical robot (Intuitive Inc, Sunnyvale, CA) in 2005 was under 50,000. This number rose to over 350,000 by 2011 [1]. Due to improved patient recovery time and reduced trauma and scarring, both types of minimally invasive procedures have replaced traditional open surgeries for a large number of applications.

While RMIS procedures do provide certain advantages over traditional surgeries, this technology has certain adverse side effects. In the work by Null et al. [2], data was collected concerning the total annual physical and economic cost of medical intervention. This group reported that the total number of deaths caused by surgical error in 2003 was 32,000. These errors resulted in \$ 282 billion in extra costs and approximately 2.4 million days spent by patients in the hospital. In total this can be converted to an average of 7 surgeries per American life. Obviously there are is a high instance of dangerous errors caused by surgeons which results in an economical burden on the health care system. While there exist fewer published statistics on the matter, the use of surgical robots by unqualified surgeons has contributed to this number of surgical errors. In order to help correct the instance of surgical errors experienced during the use of surgical robots, certain training and certification methods can be enacted to evaluate the skill level of a

surgeon before they are allowed to use this technology during surgery.

In order to properly train surgeons and medical residents to a desired level of proficiency in the field of robotic surgery, several institutions have worked to develop certain training simulators. Along with the development of training simulators, there has been much work on the standardization of tasks and training curricula to get students accustomed to the foreign nature of MIS and RMIS tools. The fundamentals of laparoscopic surgery (FLS) [3] is the most widely reviewed standard for laparoscopic surgery. The Fundamentals of Robotic Surgery consortium (FRS) [4] is a similar initiative working to provide standardized RMIS accreditation. As both of these training sequences become more advanced and standardized, an important tool will be motion tracking. Most skill evaluations for laparoscopic surgery use path length and motion analysis to gauge performance. An accurate 3-dimensional, time dependent measurement of tool path is one of the most critical pieces of information for quantitatively gauging the performance of the surgeon. In their objective evaluation study, Judkins et al. performed several robotic skills tests on both expert and novice surgeons and found:

“This study clearly demonstrated the ability of objective kinematic measures to distinguish between novice and expert performance and training effects in the performance of robotic surgical training tasks.” [5]

Information such as overall path length and task completion time has been shown to discriminate surgeons based on their level of education [6]. This work will discuss the modules currently available for training surgeons in laparoscopic and robotic skills. It will also review the various metrics used to evaluate performance within the training modules. Then, various technologies for tracking the motion of surgical tools and end effectors will be presented. A discussion of the accuracy, construction and applicability of each tracking technology to robotic systems will be presented. Finally, previous work on the subject of anatomy tracking will be discussed and the most promising technologies will be identified.

2.2 Training Modules

The development of training modules and subsequently, metrics for quantifying performance in training, has been the subject of many research projects over the past decade.

Currently, there are four available means for training surgeons to use MIS systems, the first being the physical task trainer.



Figure 2.1: The FLS Box Trainer. [7]

One class of these include ‘box’ trainers (Fig. 2.1) [7] and small-scale anatomy trainers. Originally adopted for the training of laparoscopic tools, the box trainer is made up of a shell into which surgical tools can be inserted. Inside the shell various tasks can be set up and performed by the student using a viewing monitor. Typical box training systems include tasks like peg transfer, pattern cutting or suturing tasks which target general psychomotor skills. Currently available systems include the McGill Inanimate System for Training and Evaluation of Laparoscopic Skills (MISTELS) [8] and the Advanced Dundee Endoscopic Psychomotor Tester (ADEPT) [9]. The more realistic ‘partial task trainer’ uses realistic models of human skin and anatomy to simulate certain procedures (Fig. 2.2). However the partial task systems are limited by the fact that they are not able to simulate the entire body, only specific regions and thus it is difficult to provide a completely realistic surgical set up.



Figure 2.2: A Partial Human Anatomy Trainer. Source: Simulab Corporation (Seattle, WA)

Work is currently underway by various research groups to develop standardized physical trainers for robotic surgical training, notably the FRS consortium has specified several of the requirements for a successful robotic task trainer. As presented at the 2012 Society of Laparoendoscopic Surgeons (SLS) conference, the FRS didactic and knowledge group decided on a task list including: ring tower transfer, docking and instrument insertion and clover-leaf dissection, among others [1]. Although the design details for this training and evaluation module are still under development, the pedagogical goals are mostly in place and provide solid guidelines for the tasks required by the physical trainer.

A second class of training systems consists of virtual reality (VR) trainers, which are generally setup with mock laparoscopic tool handles for control (Fig. 2.3). The user then views a virtual world through a computer screen and uses the handles to carry out various tasks. The virtual reality systems have several advantages including the ability to portray almost any anatomical region with relative ease. Another advantage is, given the end effector movement is simulated in the virtual world, the system can inherently track the motion throughout the task. However these systems have several disadvantages, including; the lack of haptic feedback, thus diminishing the realism, an increase in cost (in comparison to physical trainers), and the need for additional validation studies. For laparoscopic surgery, the most widely researched virtual reality trainers are the MIST-VR [10] and LapSim (Surgical Science, Gothenburg, Sweden)

[11].



Figure 2.3: A virtual reality laparoscopic trainer. [7]

The physical and VR laparoscopic trainers are often combined into a third trainer class, referred to as a hybrid module. In a hybrid setup the student uses replicated control systems in conjunction with fabricated anatomical shells but the process is carried out in a virtual reality setting. The added benefit to this setup is that the synthetic anatomy provides “accurate deformation and force feedback during manipulation.” [12]. One such commercially available hybrid device is the ProMIS (CAEHealthcare, Sarasota,FL). The hybrid simulators provide an increased sense of realism, however they are generally limited to specific parts of the body. The hybrid devices also add additional cost relative to pure virtual systems given the additional hardware.

For robotic surgery, a review of the literature uncovered three main da Vinci simulation devices. The first simulator is the Mimic dv-Trainer (MdVT). This project was initially undertaken at Mimic Technologies (Mimic Technologies Inc. Seattle, WA) by Jeff Berkley [13]. This simulator resembles the look and feel of the da Vinci console, incorporating endowrists, foot pedals, and stereoscopic display (Fig. 2.4). The MdVT also includes several training tasks commonly associated with surgical robotics programs. Some validation studies have been conducted on the MdVT and at least one found that the MdVT can improve performance on certain pick and place tasks [14]. Many of the simulation tasks were adopted by Intuitive Surgical Inc. for their own in-house da Vinci Skills Simulator (dVSS) device [15]. As expected, both the MdVT and the dVSS only work with the da Vinci system and due to the closed-source nature

of Intuitive, this is not an ideal module for developing a standardized training curricula.

The second simulator was developed by L. W. Sun et al. in 2007 in Hong Kong. Their work includes modeling the kinematics of Da Vinci, designing an interface that mimics the da Vinci controls and designing simulation tasks [16]. Their work primarily focused on the hardware and software to simulate the da Vinci kinematics, while the only task development that was done was simulating a bean drop using the simulator.

The third simulator is the Robotic Surgical Simulator (RoSS) developed by Ankur Baheti et al. at Buffalo, NY [17]. The initial simulator used the Omni controller from Sensable (Geomagic, Sensable Group, Wilmington, MA) and rendered a computer simulation world using C++ and OpenGL. While their graphics were relatively simplistic, their group spent some additional time considering and developing various tasks such as transferring rings from one post to another and cutting veins - all in a virtual setting. The RoSS is now commercially available from Simulated Surgical Systems (Simulated Surgical, Williamsville, NY) and has undergone more extensive validation studies. A benefit of the initial RoSS development is the modeling program uses Denavit-Hartenberg [18] parameters for the inverse kinematics of the da Vinci and so theoretically it could be replaced with the appropriate values for another robot if the need arises.



Figure 2.4: The MdVT virtual reality da Vinci simulator. Source: Mimic Technologies (Seattle, WA)

The final class of training module available is also the most obvious, using a real tool in the operating room on the actual patient. This situation works for both the

laparoscopic and robotic tools and is inherently the most realistic training available. However, the downfalls of allowing an inexperienced surgeon to use these devices on a patient are as equally as obvious. The chance of a novice surgeon causing an accident with a laparoscopic tool is much greater and thus poses an increased risk to the patient's health. Given the complex and powerful nature of robotic systems in particular, even slight mishaps can be life threatening to the patient. Even if mishaps are avoided, the lack of experience that a novice has with complicated MIS procedures will typically lead to increased operating room (OR) times, a general increase in recovery time and ultimately a greater financial burden. For these reasons, the first introduction to MIS and RMIS methods requires an offline training system. Yet even while simulators and other training devices are developed, real-world exposures to these tools will always be the final pedagogical step for laparoscopic and robotic training and therefore tracking systems will still be useful for gauging real world performance.

Currently the established training and certification tasks for laparoscopic surgery include the FLS peg transfer, pattern cutting, ligating loop and suturing with an extracorporeal knot [19]. The FLS tasks were originally determined by SAGES in the early 2000's, as such these tasks have been the subject of many validation studies. As determined by the FRS consortium, the tasks for robotic surgery certification will include ring tower transfer, docking and instrument insertion, knot tying, cloverleaf dissection, vessel energy dissection, railroad track suturing and 4th arm cutting [1]. These tasks will be performed on an actual RMIS device in conjunction with a physical training module containing the required task setups (Fig. 2.5). In addition some promise has been shown in the use of virtual reality simulators for the da Vinci [20]. It is very likely that robotic VR simulators will also play a role in FRS training and certification. It is worthwhile to note that for all non-VR robotic training modalities, an actual da Vinci or similar robot is required to complete the task.

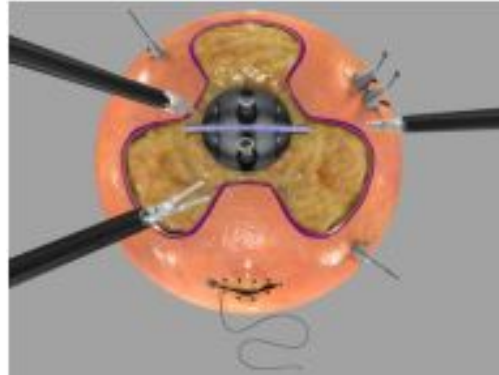


Figure 2.5: FRS Certification Module Prototype. [1]

There are multiple schools of thought concerning the proper metrics that should be analyzed when quantifying psychomotor skills, a primary one being dexterity analysis [21]. Dexterity analysis attempts to quantify performance in a certain task by determining the number of movements required to complete the task. The accuracy of the movement measurement depends on the scale of the task in question. The FLS has clearly defined their basic performance metrics for laparoscopic surgery by primarily considering dexterity analysis. For robotic surgery these metrics are not as clearly defined but given the tasks proposed by the FRS consortium, it is assumed many of the metrics will carry over, particularly those relating to completion time and motion dexterity. As described by [12], the primary laparoscopic training metrics include but are not limited to:

- Path Length - The cumulative 3 dimensional path length traversed by the end effector.

$$P = \sum_{N=1}^{N=F} \sqrt{(X^N - X^{N-1})^2 + (Y^N - Y^{N-1})^2 + (Z^N - Z^{N-1})^2} \quad (2.1)$$

(Where N is a discrete time step)

- Economy of Motion (EOM) - Extent of the deviation from an ideal path in 3-dimensional space. The total error from the direct path between the start and end points of a task. This can also be calculated as a temporal deviation from an ideal task completion time.

$$E = \int_0^F \sqrt{(X_i(t) - X(t))^2 + (Y_i(t) - Y(t))^2 + (Z_i(t) - Z(t))^2} dt \quad (2.2)$$

(Where (X_i, Y_i, Z_i) is an ideal position)

- Time - The total time required to complete a task.
- Motion Smoothness - A metric describing sharp changes in acceleration. This can be calculated by looking for instances of large jerk values (J).

$$J = \frac{d^3}{dt^3} P(X, Y, Z) \quad (2.3)$$

- Response Orientation - An amount which the end effector rotates about the 'roll' or tool axis.

$$R = \sum_{i=0}^{i=F} \frac{d}{dt} Z_i \quad (2.4)$$

(Where Z represents the orientation on the end effector)

Figure 2.6: Equations for motion metrics.

The equations shown above describe the training metrics mathematically in 3-Dimensional Cartesian coordinates. These equations indicate the relationship between the metrics and accurate temporal and spatial data.

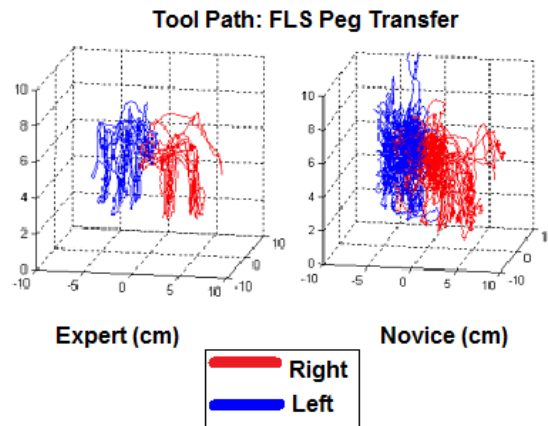


Figure 2.7: Laparoscopic Tool Path for an Expert (left) and Novice (right). [22]

Figure 2.7 demonstrates the increased accuracy and efficiency that an expert surgeon has over laparoscopic tools as opposed to a novice surgeon. While not a definitive source of certification, this suggests that a seasoned expert employs a much more ideal tool path, indicating that economy of motion can signify a surgeon's ability level. As seen in the above equations, each of these performance metrics requires both accurate temporal and spatial data to be calculated. In addition to these dexterity measures, other visual-spatial skills are considered, such as depth perception and grasping proficiency. These additional skills are slightly harder to quantify. Furthermore the FRS curriculum [1] requires some way of quantifying performance or skill development in the following areas:

- 3-Dimensional Imaging - Relating 3-D visuals with required movements.
- Master Slave Relationship - Translating Joystick control into end effector movement.
- Motion Amplification - Related to Master-Slave control but incorporating the correlation between small and large movements.

- Fulcrum Elimination - Becoming accustomed to direct motion as opposed to the fulcrum based motion in laparoscopy.

These robot-specific training measurements are not finalized since the FRS training module design is not yet complete. However given the tasks specified by the FRS, it stands to reason that tool motion, completion time, obstacle avoidance and overall accuracy will all be required by the certification criteria.

An obvious requirement for almost all psychomotor skill metrics is accurate motion analysis [21, 5]. In order to gauge path length and motion smoothness, a tracking system will need to measure the motion and position of the end effector tips to within a certain accuracy. Given the working volume of a da Vinci is quasi-hemispherical with a radius of 24cm [23], it is reasonable to require that a tool tracking method be capable of achieving sub-millimeter accuracy. For most VR simulators, such accurate tracking is inherent to the system. Yet for physical laparoscopic trainers a tracking device must be used. Although a robotic arm such as those used on the da Vinci can be localized using kinematics, these values contain inherent inaccuracies. The majority of these errors are due to finite uncertainty in the encoders found in the prismatic or evolute joints. These uncertainties propagate through each joint and when a Denavit-Hartenberg table is used to calculate end effector position, this uncertainty will be exacerbated. Recent testing has found the da Vinci can carry a physical fiducial localization error as large as 1.3mm [23].

In addition to the joint localization error, the compliance in the robotic tool arms and cables can further amplify position measurement error. As identified in [23], it is reasonable to assume that the position error due to compliance in the tools is non-negligible, hence even with perfect joint encoders, the tool tip position would still not be accurate. This is in comparison to visual tracking systems such as the Optotrak 3020 (Northern Digital Instruments, Waterloo, ON, Canada), which has a localization error of only 0.25mm [23]. Although Reiley et al. [24] discussed the addition of sensors to robotic end effectors for tracking motion; they stipulated that it would be difficult given the embedding requirement and sterilization issues. Therefore when using surgical robots, this secondary tracking method should ideally involve minimal or non-existent sensor placement on the end effector.

The two broad areas of tracking that will be discussed are tool tracking and anatomy

tracking. The tracking of anatomy is concerned with creating a 3-dimensional model of the surrounding tissue and vessels near the surgical instruments, registered to a given world coordinate system. The goal of tool tracking is an accurate localization of the laparoscopic tool or end effector within the same coordinate system. More specifically, the desired information includes 3-dimensional position of the end effector relative to a reference frame such as the camera or a fixed world reference frame. Additional information such as the orientation of the specific tool in use is also required. As seen in Figure 2.8, the object to be tracked would ideally be the center of the manipulator wrist or jaw axis (coaxial with label 60). Then an accurate representation of the 3 Degree of Freedom (DOF) orientation of the wrist is also of great interest, although little research exists regarding the skill discriminating power or wrist orientation information.

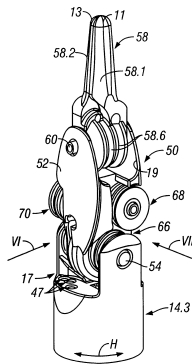


Figure 2.8: Standard end effector. [25]

Tool tracking methods can be broken down into two fundamental groups: Active tracking and Passive tracking. Active tracking involves placing a photo or electromechanical device on the tool end in order to send a signal to a nearby sensor. This emitter on the tool can either be an LED in the case of vision-based systems, or magnetic transmitters in the case of electromagnetic systems. Conversely, passive tracking uses only a sensor at some distance away from the end effectors which tracks position using some inherent property of the tool such as a physical model.

Tracking systems can be broken down further into the various technologies they employ. For simplicity this paper will consider auditory, visual, mechanical and electromagnetic techniques. Each category has various benefits and drawbacks, which will be

considered.

Several devices already exist for the purpose of tracking laparoscopic tools both in simulation and in the operating room. Although these exact technologies cannot be transferred to the tracking of the robotic end effectors, they serve as a basis for working toward robotic end effector tracking. As mentioned in their overview of the existing laparoscopic tracking methods available in 2007, the Chmarra group at Delft university stipulated that the primary features which should be considered while evaluating a tracking system are the type of system (passive or active), the type of mechanism, the degrees of freedom which can be tracked, the applicability to (OR) procedures, the availability of haptic feedback, functionality with real laparoscopic tools, and the accuracy of the tracking [26]. This paper will consider a subset of these features including system type, mechanism and accuracy with an additional focus on transferability to robotic systems and cost. The systems considered include both commercially available products as well as setups developed at research institutions.

2.2.1 Mechanical Systems

A primary mechanical device currently in place for tracking laparoscopic tools is the gimbal mechanism coupled with either optical encoders or potentiometers. According to older patents, a gimbal is described as

“a mechanism allowing controllable reorientation of an article about a spatial point. The point may be selected as the center of mass of the article to that only angular momentum changes are involved in reorientation.” [27].

The mechanism can then be connected to either a potentiometer or an optical sensor in order to calculate and monitor changes in reorientation. These reorientations are then simply extrapolated down the length of the tool shaft to determine the position of the tool tip.

One of the current tracking systems that uses a mechanical device is the EDGE Trainer (Simulab Coporation, Seattle, WA). This device is the commercialized evolution of the BlueDragon and RedDragon systems (Fig. 2.9), originally developed at the University of Washington BioRobotics Lab, Seattle. The newest EDGE system utilizes a three-axis, remote center, spherical mechanism connected to potentiometers and optical

encoders in order to accurately track position, orientation, and applied force. This device can either be connected to a training simulator or potentially utilized in the operating room to track surgeons' movements, at least on precise models during training.

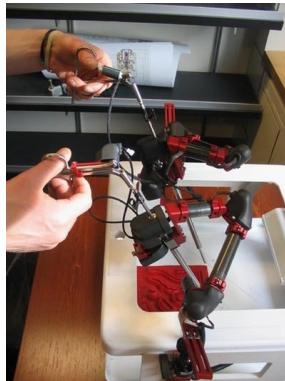


Figure 2.9: Tracking mechanism on the RedDragon. Source: UW Today (University of Washington, Seattle, WA)

There also exists a prototype tracking system developed at the Delft University of Technology known as the TrEndo. This device also used a gimbal mechanism whose motion is then measured by optical sensors. Since the da Vinci robot already uses its own kinematics and mechanical design to infer position and orientation, it will not be useful to incorporate any other mechanism such as the gimbal. However, this prior work is important to understanding the evolution of tracking systems.

2.2.2 Incorporated Simulator Systems

Many companies who produce and develop physical and virtual reality simulators for laparoscopic tools also develop tracking systems for these devices. The reason these simulators require tracking systems, is so the student can use either real laparoscopic tools or replicas to control the simulator. Then the motion of the end effector is captured and repeated in the simulation. This tracking hardware functions more as a control device or a joystick. Since this work is more concerned with robust tracking systems and those which can transfer to robotic surgery, the discussion of these systems will be brief. Xitact ITP (Mentice AB, Gothenburg, Sweden) is one such product which uses optical sensors and encoders to track tool motion (Fig. 2.10). The IOMaster series

(Research Center Karlsruhe, Karlsruhe, Germany) is another line of control stations for laparoscopic simulators. The IOMaster utilizes a combination of optical encoders and Hall effect sensors to track position and orientation.



Figure 2.10: Tracking mechanism on the Xitact ITP. Source: Mentice AB (Gothenburg, Sweden)

Immersion Inc. © (San Jose, CA) has developed a range of hardware interfaces for the specific purpose of providing active tracking mechanisms to control laparoscopic simulators. Both the Laparoscopic Surgical Workstation (LSW) and the Virtual Laparoscopic Interface (VLI) utilize gimbal mechanisms in conjunction with electromechanical transducers to track position and orientation with 4 DOF. Their other device, the Laparoscopic Impulse Engine incorporates servo-motor actuators to map the motion. Although this device can use real laparoscopic tools, it can only be used in simulation. The LSW can achieve resolutions of around $8 \mu\text{m}$ and while resolution does not explicitly infer accuracy, this number provides a relative scale. Immersion's hardware is utilized in the package simulators produced by other companies such as the Computer Enhanced Laparoscopic Training Systems (CELTs) developed by the simulation group at the Center for Integration of Medicine and Innovative Technology (Boston, MA).

Another simulator with utilizes tool tip tracking is the PromIS augmented reality simulator (CAE Healthcare, Sarasota, FL). By wrapping strips of yellow tape near the end of laparoscopic instruments, a set of three cameras is able to use color tracking to determine position and then triangulate the depth to the tool (Fig. 2.11). The accuracy of this tracking system is not readily published, but contact with the company led

to an answer of around 2mm accuracy, typical for optical tracking systems. However, this device can only be used in simulation and not in the operating room. The Advanced Dundee Endoscopic Psychomotor Tester (ADEPT), developed at the University of Dundee, Dundee, Scotland, is another simulator that comes with tracking technology. In this case laparoscopic instruments are inserted through a gimbal device and the motion is then measured using a potentiometer in the gimbal [28]. This device can obtain accuracies around 0.5mm when measuring 300mm from the pivot point [26].



Figure 2.11: Tool tip tracking on the ProMIS system. Source: CAE Healthcare (Sarasota, FL)

Also available is the Simendo simulator (Simendo, Rotterdam, the Netherlands). This laparoscopic simulator also uses a gimbal device to hold the instruments, however instead of a potentiometer, this device utilizes optical sensors to track gimbal movements and measure position.

The majority of the incorporated simulator systems are mechanical, therefore the only likely technology that may transfer to robotic applications is the optical tracking found in the ProMIS. Although the ProMIS itself is very expensive simulator, it is likely that this method of optical tracking can be inexpensive to implement. It is useful to note that mechanical tool tracking systems such as the Xitact ITP system typically have prices in the range of \$24,000 - \$30,000 but can vary significantly based on the accuracy. A large number of these companies do not publish accuracy information for their tracking systems, however given the ADEPT interface accuracy (0.5mm) it is safe to assume other systems have similar accuracies.

2.2.3 Ultrasound Systems

Another, subset of tracking systems technology is ultrasound and auditory technology. Using a combination of ultrasound transmitters and receivers, an external device can track the tool tips inside either a simulator or a patient, given the transmitters are situated on the tools inside. This idea was initially proposed in 1991 by Morris et al [29] for the tracking of generic robotic end effectors. They were able to get a rudimentary accuracy of $\pm 3\text{mm}$.

A group at the Delft University of Technology has introduced a prototype device using this method. Their prototype work can track the 3-dimensional position of a surgical tool with an accuracy around $80\ \mu\text{m}$. This setup uses the time of flight method by sending out a two-frequency, ultrasound burst [30].

The method of ultrasound tracking has been commercialized by the German company, Zebris Medical (Zebris, GmbH, Aachen, Germany). Using a modified Zebris CMS ultrasound system, Sokollik et al. were able to track laparoscopic instrument movements in 3-dimensions [31]. The Sokollik group was more concerned with using this tracking system to assess training and skill and therefore did not specifically discuss the accuracy of their system. However, the most current iteration of the Zebris CMS system, the CMS-HS (Fig. 2.12) is reported by the company to have a tracking error between 1.15 - 1.35mm [32]. The cost associated with this system was quoted by the company to be between \$25,000 and \$30,000.



Figure 2.12: The Zebris CMS-HS ultrasound tracking system. Source: Zebris GmbH (Aachen, Germany)

Given the relative ease with which these ultrasound transmitters can be transferred to new tools and the fact that the transmitters can be sterilized, this should be considered one of the more viable options currently available for robotic tool tracking. The primary downside of ultrasound tracking is the cost and typically wired connection to the tool tip is required.

2.2.4 Visual Systems

Another development from the laparoscopic tool tracking realm which has a high likelihood of working with a robotic system is optical tracking. Optical tracking utilizes a video feed with the surgical tools in the field of view. Each individual frame of the video feed is then analyzed using a computer vision based algorithm to find the 2-D pixel location of the tool. These systems typically come in two modalities. The first type, the active version utilizes an infrared (IR) LED beacon placed on or near the tool end which is then picked up by three CCD cameras, situated around the tool. These cameras can then triangulate the position of the tool in some world reference frame. The second type, the passive version instead uses reflective IR spheres placed on the tool ends and instead the cameras emit the required light and the spheres simply reflect the light to be captured by the cameras. The cameras can be individually purchased from companies such as Immersion SAS (Bordeaux, France), who developed the ARTrack2 © camera for IR tracking and CYCLOPS camera for stereo tracking.

Other companies commercially provide the entire optical tracking system. Two such companies are Image Guided Technologies (Boulder, Colorado) which has now been acquired by Stryker Inc. (Kalamazoo, MI) and Northern Digital Inc. (Ontario, Canada). Northern Digital has two products currently in place, the Optotrak and the Polaris © (Fig. 2.13). The Polaris comes in both an active and a passive model, while the Optotrak comes only in an active setup. The form factor of the Polaris Vicra © system is under 10in long, making it a much more feasible option than the larger Spectra © version or the 1m long Optotrak system. The Polaris system also has the option of using wireless marker spheres. Image Guided Technologies most recently produced the Flashpoint Camera system. This came in multiple models which functioned at different distances (300mm, 580mm or 1m) and also came in either the passive or active model.



Figure 2.13: The various Polaris systems. Source: Northern Digital Inc. (Ontario, Canada)

As shown by Kwartowitz et al. [23] the Optotrak 3020 can obtain accuracies up to 0.1mm for static positioning. Khadem et al was able to show that the FlashPoint Camera had jitter ranges of $0.028 \pm 0.012\text{mm}$ for the 300mm model, $0.051 \pm 0.038\text{mm}$ for the 580mm model and $0.059 \pm 0.047\text{mm}$ for the 1m model. They also found the Polaris camera had jitter ranges of $0.058 \pm 0.037\text{mm}$ for the active and $0.115 \pm 0.075\text{mm}$ for the passive [33]. In their paper, ‘jitter’ refers to a deviation of repeated measurements from an average. This is merely a more specific version of accuracy. While this accuracy data is obviously for previous iterations of these systems, these data still provides a reasonable idea of the accuracies that can be obtained by optical systems.

The recent acquisition of Image Guided Technologies by Stryker Inc. has made all current pricing data for the Flashpoint system unavailable. However, for the Polaris systems, the current pricing for the smaller Vicra © is around \$12,000 while the larger Spectra © is closer to \$23,000. It is worthwhile to note that the current Optotrak Certrus © system costs over \$50,000.

Several companies including Stryker Inc. and others produce high definition surgical camera devices that could theoretically incorporate computer vision algorithms for tracking tools, however these systems do not explicitly incorporate tracking methodologies and therefore were not considered in the scope of this paper.



Figure 2.14: The Polaris Vicra system in surgery. [34]

Several research groups around the world have made great advancements in the field of optical tracking with applications to surgical tools. The types of algorithms used to perform this optical tracking falls into 4 main categories: color based or fiducial marker tracking, color segmentation of the tool versus background, geometric constraints including line detection, and model based feature extraction. Each method of tracking has certain advantages with subsequent trade offs. The primary discrepancy exists between accuracy and computational time.

One of the easiest and most obvious methods for tracking the tip of the surgical tool is to place an easily identifiable marker on the tip of the tool and then search each individual frame for that marker(Fig. 2.15). In the case of Wei et al. [35] typical frames from surgical videos were analyzed to find a color that rarely appeared in the surgical environment. This color was then placed on a small plastic ring and affixed to the end of the tool shaft. The computer program then searched each frame for a large area consisting of that particular green color. Using this method, the group was able to achieve a frame rate of 17 Hz at a resolution of 512x512. Krupa et al. [36] used 3 collinear LEDs affixed on the tip of a custom surgical tool and subsequent color tracking to identify the location of the tools in the individual frames. The displacement between the 3 LEDs can then be used to estimate orientation and depth. This group reported a computational performance of 50 hz (FPS), which is much higher than the capture of rate of typical video sources. However, it should be noted that this requires an active tool marker configuration and cannot be achieved passively. Groeger et al.

[37] implemented a similar method but instead utilizing a blue marker tip and adjusting their color segmentation algorithm accordingly. Their group was able to achieve a similar performance of around 15-17 Hz. The primary issue with this method is the alteration of the surgical tool. This requires the sterilization of the tool tip and the marker and does not permit the use of any tool, agnostic to the use of marker affixing.



Figure 2.15: Tracking a surgical tool via a blue marker.

A second method implemented by several researchers for finding the tool tip location is the use of color segmentation over the entire image (Fig. 2.16). Color segmentation is an attempt to characterize individual pixels as belonging to a certain range in color spectrum. In this case this includes either the anatomy (background) or the tool shaft. Several methods exist for performing color segmentation, the most popular of which is Otsu's method [38]. Lee and Uecker [39] developed a library of pixel values to distinguish between anatomy and tools. Then using a Bayesian filter they attempted to label each pixel in a given frame as one or the other. Once the requisite tool pixels were labeled, they attempted to fit a bounding rectangle around those pixels to model the tool shaft position. Using the available computational power this group was not able to perform this tracking in real time and were performing offline calculations. Doignon et al. used a similar method to characterize individual pixels as either anatomy or tools ([40], [41]). Their group utilized the Hue-Saturation-Luminance color representation and scanned each frame to search for pixels with a saturation value and hue value in a particular range so as to classify it a tool pixel. Once the tool pixels were identified they used a geometric constraint to fit lines to the pixels and locate the tip. This

algorithm achieved 16 FPS performance but lacked 3-Dimensional extraction at the time of publication.

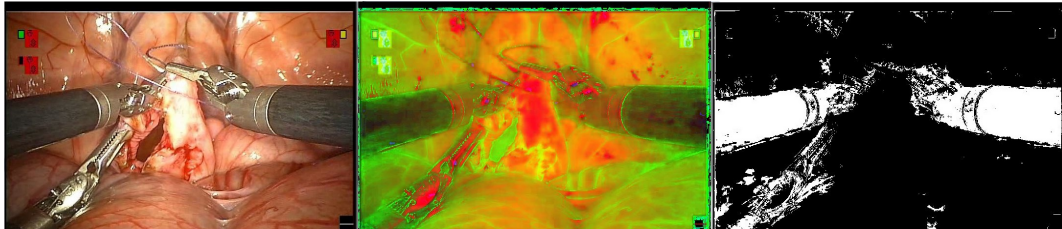


Figure 2.16: Tracking a surgical tool by segmenting image color.

The use of color markers or pixel segmentation has the benefit of near real time performance. However the standard problem with color characterization is that it is difficult to maintain robustness in varying conditions such as lighting changes, specular reflectance and color saturation. Each group that utilized these methods reported significant issues related to robustness in varied conditions. Once solution to these issues in robustness is to instead use a purely geometric representation of the tool and search for an instance of that geometric constraint in an individual frame (Fig. 2.17).

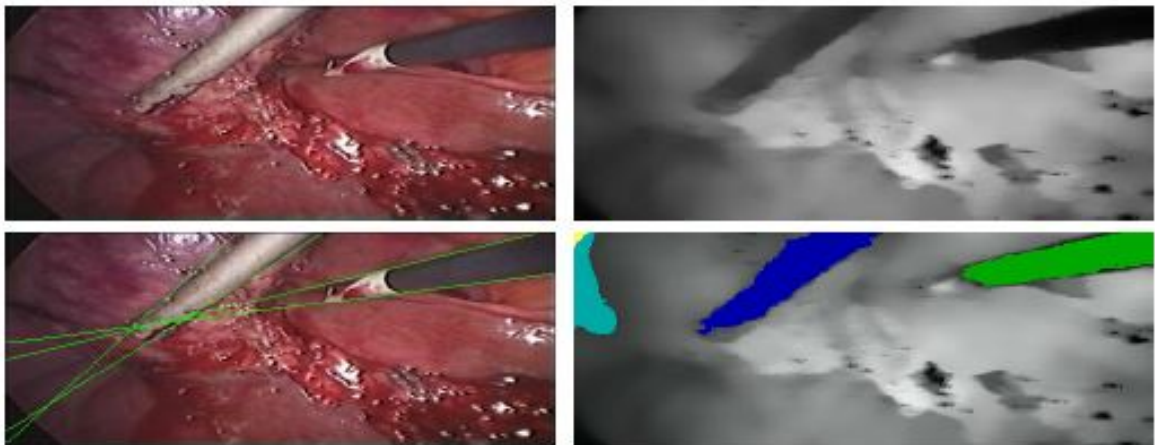


Figure 2.17: Tracking a surgical tool using geometric constraints. [42]

An early implementation of the geometric constraint method was pursued by Voros

et al. [43]. Their group used as their primary constraint the insertion point of the laparoscopic tool in the abdomen. This location has to be physically measured by the surgeon. Then by using an edge detection algorithm, the pixels are segmented into strong edges. The pixels which meet the constraint of originating from the insertion point are then grouped and a hough transform is applied. The hough transform uses a voting process to determine points which lie on a line described by the parameters ρ and θ [44]. The Voros group used the gradient of the edge detection pixels to find points with opposite gradients to identify edges on opposite sides of the tool shaft. These lines are then used to compute a center line traveling down the shaft. The pixels underneath this centerline were then further characterized using Otsu color thresholding to gauge whether they belonged to the tool or the background anatomy. The pixel where this segmentation switched is then used as the tool tip location. While this group did not achieve 3-Dimensional tracking they were able to achieve accuracy of 11 pixels in 87% of the frames (in 2 dimensions) with a computational performance of 10 Hz (FPS) at a 200 x 100 resolution. Unfortunately this group also suffered from specular reflectance issues while using color segmentation.

A similar approach using the geometric constraint approach was pursued by Doignon et al. [42]. This group also used the insertion point as a geometric constraint and then performed color segmentation on gray regions, followed by a hough transform. This group did not accurately locate the tool tip location, merely the boundary lines of the tool shaft and did not provide computational performance metrics.

More recently Wolf et al. [45] used similar geometric constraint method but also included a Conditional Density Propagation method based on a Geodisc model. This method also relies on the insertion point of the tool, but automates the calculation of the insertion point using a calibration method. The Geodisc model is fitted within the constraints of the insertion point and hough transform lines. Not including the hough transform time, this groups fitting method achieved between 8-16 Hz (FPS) in performance time.

The use of geometric constraints and fixed location information has several benefits including robustness to lighting changes and a confined search space. However, several of these attempts still employed the use of color thresholding eventually, thus incurring the issues therein. The other issue with the use of geometric constraints is the

required measurement and localization of the constraint (in most cases the insertion point). However the geometric methods discussed all seem to be successful in terms of computational time and reasonable accuracy. They also employ the use of the hough transform indicating that algorithm as a useful tool.

The final and most sophisticated method for tracking surgical tools is the model-base feature extraction method. model-based feature extraction first requires the researcher to build a complicated and large library of pixelized representations of the tool tip to be tracked. This involves obtaining a large number of images depicting the tool in every possible orientation and pose and then converting those images into a library definition of the tool. This library typically takes the form of Haar features or a boosted cascade of simple features [46]. This library can then be queried in each frame to locate the tool (Fig. 2.18).

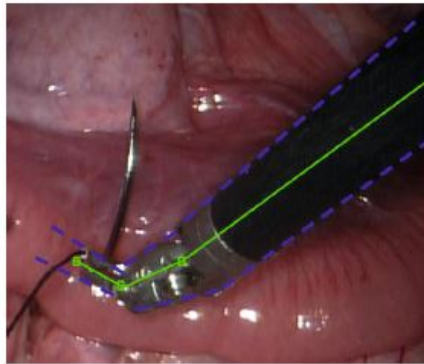


Figure 2.18: Tracking a surgical tool using a known physical model. Source: [47]

One implementation of the model-based tool detection was pursued by Pezzementi et al. [48]. This method involved training a model using Computer Aided Design (CAD) models and rendering these models in the OpenGL graphics library [49]. These CAD models were then used a classifier for detecting the tool tip in each frame. This group lists the computational performance for this algorithm at about 800 ms per frame and therefore cannot be implemented in real time.

A second implementation of model-based feature detection was done by Reiter et al. [47]. This group also used CAD models to build their library classifiers and stereo correspondence to achieve 3-dimensional tracking. This group achieves very accurate

and sophisticated tracking, not just of the tool tip location but also the configuration of the end effector (wrist) joint. They report an accuracy of 86% but with a computational time of 1.2 seconds per frame and therefore cannot be used for real time tracking.

While the model based feature extraction method provides excellent accuracy and the additional benefit of end effector configuration data a review of the literature also indicates that the computational burden of a such a sophisticated algorithm hinders its ability to be applied to real time tracking. However this method will likely become more plausible as computational power grows.

Given the speed (a maximum of 15-17 Hz [37]), sufficient accuracies, and relatively low cost of vision systems, this tracking method seems very useful in tracking surgical robotic tools. A review of the literature indicates that color characterization is fast but not sufficiently robust while full feature extraction is robust but relatively slow. The primary issue that all optical tracking systems suffer from is the line of sight requirement. If a surgeon using the da Vinci suddenly positioned the end effector so it was occluded from all cameras, the position would be lost. However it seems that for all other purposes a robust, fast and accurate vision algorithm could be developed by using the successful elements found in previous attempts and combining them.

2.2.5 Electromagnetic Systems

The electromagnetic transmitter and receiver setup allows for relatively small transmitters to be placed on tool tips which then emit magnetic fields. The magnetic fields are detected by magnetic sensing coils inside the receiver, the receiver is placed outside the body.

Several companies develop this type of tracking device. Ascension Technology Corporation (Milton, VT) has produced several electromagnetic tracking systems such as trakSTAR © and driveBAY ©. This company is now part of Northern Digital. Their devices can achieve very fast update rates (around 300 updates per second) and can track 6 DOF while using multiple transmitters. The typical errors obtained for this system are around 4.0mm, even using their flat transmitter which overcomes distortion produced by the operating table [50]. Polhemus © (Colchester, VT) has developed both the Patriot and Fastrak (Fig. 2.19 systems for 6 DOF tracking. The Patriot is available both as a wireless and wired setup and can obtain resolutions of up to 0.01 mm

(in ideal conditions). Such a system costs around \$5,900 and also suffers from magnetic field distortion effects. These are difficult to overcome in the operating room or near a robot given the large amounts of ferrous metals.



Figure 2.19: The Polhemus Fastrak system. Source: Polhemus (Colchester, VT)

Another electromagnetic tracking system is the Aurora © also from Northern Digital. The system utilizes a table-top field generator which is placed underneath a patient, this tracks small sensors with sizes near 10mm. The system has a 6 DOF accuracy around 0.48mm. The base price for the Aurora system is around \$15,000 with an increased price for the table-top OR generator. The primary downside of electromagnetic tracking is that ferromagnetic objects near the region to be tracked can cause disturbances in the magnetic field, which result in tracking errors of up to several millimeters.

Solutions have been proposed to deal with the shortcomings of electromagnetic setups. The work of Feuerstein et al. [50] resulted in a hybrid optical-magnetic system with the goal of redundantly checking for errors in the electromagnetic sensor by using the ARTrack2 optical system. They were able to minimize tracking errors to around 3.15mm by using this setup in the presence of metal. However the metal situated in the robotic arms of the da Vinci and other systems will likely still pose an issue. Northern Digital has also done some research into mapping the distortions in the magnetic field due to stationary metallic objects and subsequently compensating for these distortions [51]. However their work is not yet integrated into their systems.

2.2.6 Comparison of Tool Tracking Technologies and Gap Analysis

It is worthwhile to consider all the available options for tool tracking for minimally invasive applications. Although several options may not be viable, they demonstrate the core technologies currently available. The most promising technologies with cross-applicability to robotic surgery seem to be optical and auditory systems. The optical systems use minimal sensing technology and don't require wired connections to the tools. The auditory systems also require smaller sensors on the tools and are not susceptible to line of sight requirements. Conversely, mechanical tracking systems are already incorporated into a da Vinci arm and extra tracking mechanisms would add bulk to the robotic arms. The electromagnetic system's accuracy would suffer greatly from distortions due to the metal material in the robotic arm itself. Therefore the mechanical and electromagnetic technologies are not as feasible.

The available technologies come in a range of price points. The systems using mechanical devices such as Xitact ITP or Trendo cost in the range of \$24,000- \$30,000 with more sophisticated system such as EDGE costing more. The optical systems, such as the Polaris ©, cost between \$12,000 and \$23,000. The auditory-ultrasound systems, such as the Zebris products, cost between \$25,000 and \$30,000. Finally the electromagnetic systems such as the Aurora © cost around \$15,000. The integrated simulator-tracking systems such as LapSim carry additional costs given the simulation hardware involved, which is unnecessary for this work. However, it is reasonable to assume that the fundamental optical tracking system in the LapSim device could translate well to robotic applications. Using products such as the Sony © high-resolution ccd camera (\$448.00), it would be possible to reverse engineer the LapSim optical tracking system. Similarly, all these prices reflect video-based, commercially available products and therefore a custom design could cost less.

Between optical and ultrasound systems, it seems that video-based optical systems are likely the most cost efficient, although it is difficult to retrieve a complete list of cost figures for the various systems listed. It is also a more feasible task to develop a custom video-based optical tracking system than an ultrasound system because of the availability of inexpensive high resolution cameras and powerful open source programming libraries for computer vision, such as OpenCV [52]. Table 2.1 summaries the various benefits and drawbacks for each system type. This table also includes relative

cost figures and applicability to robotic systems.

Table 2.1: A comparison of the various tracking technologies currently available for laparoscopic tools. Companies include major developers both present and past. Cost is given in a range dictated by the prices that were publicly available.

Technology	Examples	Benefits	Drawbacks	Accuracy	Cost	Robot Use
Mechanical	Xitact ITP, EDGE, TrEndo, IoMaster, Simendo	Several producers, relatively simple technology	Less accurate, bulky hardware	$\approx 0.5\text{mm}$	\$24,000 - \$30,000	No
Optical	Polaris, Optotrak, ProMis, Research Labs	Minimal sensors, wireless connections	Tool Augmentation	0.028 - 0.11mm	\$12,000 - \$23,000	No
Computer Vision	[35] - [48]	Software based, minimal hardware	Line of sight requirements	1 - 2mm	N/A	Yes
Ultrasound	Delft University Prototype, Zebris CMS-HS	Minimal sensing and no line of sight requirements	Increased cost, more complex custom system	$80\ \mu\text{m}$ - 1.35mm	\$25,000 - \$30,000 (estimate)	Yes
Electromagnetic	trakSTAR, driveBAY, Patriot, Fastrak, Aurora	Relatively inexpensive multiple producers	Distortions in magnetic fields due to nearby metal	0.48 - 3mm	\$15,000+	No

2.3 Anatomy Tracking

The tracking of anatomy would have several benefits for RMIS techniques and health care in general in addition to potential use in training routines. Anatomy tracking could be used in real surgeries to help model patient tissue and register target sites, indicated through MRI, CT or other diagnostic tools. This type of tracking could also be used to employ virtual fixtures or context-dependent control algorithms in the robot arm to help

prevent tissue injury. Motion compensation routines could also be used in surgery to perform delicate procedures around organs with breathing or heartbeat induced motion. The potential applications of accurate tissue and anatomy tracking are extensive.

The tracking of anatomy raises additional challenges in comparison to tool tracking due the 3-Dimensional nature of not only the tracking environment but the surface itself. In order to track the entire organ system relative to a surgical tool, a tracking system must be able to register an entire field of contour points. These points make up the surface plot of the anatomy in question. Then using these points, a matching routine can be used to develop a model of the deformable surface and relative position. While a tool tracking system must track a single point with 6 degrees of freedom, anatomy tracking requires 3-Dimensional tracking for an entire 3-D surface.

Due to the lack of wide spread commercialization in the area, anatomy tracking systems are far less common than tool tracking systems. The current options for anatomical tracking are limited to either fiber optic sensing or visual tracking of anatomy inside a patient. Since this type of tracking is still in relative infancy, this work will look at the work developed in the last decade and speculate at the available options and technologies that could be introduced.

Instead of using cardiopulmonary bypass to stop the heart for cardiac surgery, Thakral et al. [53] proposed a rudimentary method for modeling the motion of a beating chest cavity wall. In their work, a fiber optic displacement sensor with a 3 μm resolution was used to obtain distance measurements along the z-axis to a beating chest cavity of a rodent. A fiber optic displacement sensor functions by transmitting a concentrated light source (typically with a diameter around 0.2mm) from a probe onto an object. Then the intensity of light reflected back is measured by a photo-sensor and the relative drop in intensity is used to compute the distance to an object (Fig. 2.20). Unfortunately, this intensity based approach is susceptible to varying lighting conditions and is not particularly robust.

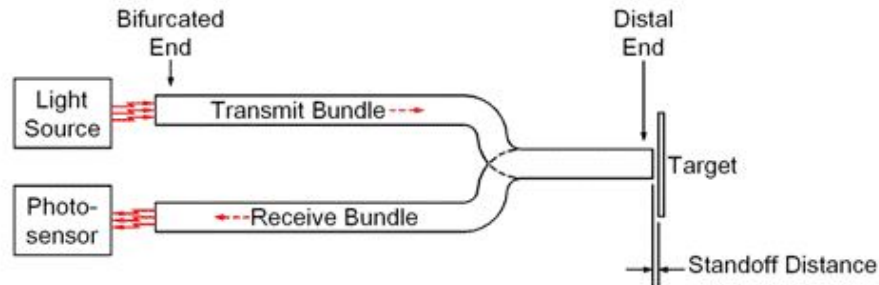


Figure 2.20: Diagram of a Fiber Optic probe. [54]

Using this distance measurement, various adaptive filtering and fourier linear combination methods were used to model the periodic nature of the physiological motion. Although this fiber optic probe only measured motion in one dimension, the process could theoretically be extrapolated into a full tissue tracker using an array of probes. The motivation behind their work was to eventually include the models of these complex biological signals in robotic actuators. The motion could then be automatically compensated so the surgeon using the robotic device would perceive no movement. This early work demonstrated that physiological motion could indeed be tracked and modeled, at least at a basic level.

Nakamura et al. [55] developed an early prototype tracking system for motion compensation during bypass surgeries. Their system utilized a color NTSC camera and a high speed monochrome camera capable of 955 frames per second (FPS) . These two cameras were situated into a half-mirror prism configuration, sharing a single zoom lens. These cameras then tracked a small yellow bead to be placed on a beating heart. They successfully performed the tracking in-vivo for a pig and were able to isolate three dominant frequencies for the motion of the heart.

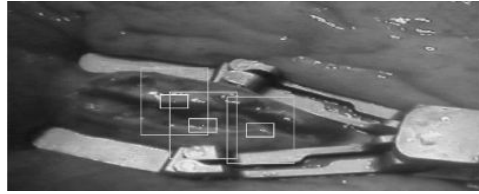


Figure 2.21: A mechanically stabilized heart with landmarks and tracking areas. [56]

Ortmaier et al. [56] also used visual tracking to perform motion estimation and compensation for beating heart surgery. Their work was done using 8-bit grayscale video from the laparoscopic cameras. Instead of tracking physical devices, the authors preferred small, natural landmarks and texture areas found on the heart (Fig. 2.21). A novel modeling and prediction method was used by incorporating Takens Theorem to predict the frequencies of motion for the heart surface trajectory. They predict that their outlier detection scheme will allow motion compensation such that robotic arms can compensate for all but 0.4mm of the heart's motion. Their method was able to perform this tracking with only an additional 0.2ms/frame penalty, albeit without the additional information provided by stereoscopic vision. Their future work will include using this algorithm to perform motion compensation with a physical surgical arm setup in real time.

Ginhoux et al. [57] were also able to perform successful visual servoing for motion minimization and compensation in heart surgery. Their work involved a 500 fps grayscale camera and a prototype surgical arm from Sinters, SA (Toulouse, France). The camera tracked a laser spot projected onto the heart. Then a Model Predictive Control method was used to translate the motion of this spot into global coordinates and then provide control increments to the robot arm so that the surgeon perceives no motion. This method was successfully used on the beating heart of a living pig without external stabilization.

Howe et al. utilized a SONOS Real-time 3D Ultrasound System (3DUS) from Phillips Medical (Andover, MA) in conjunction with several quasiperiodic predictive filters to model heart motion [58]. Typical 3DUS systems carry severe accuracy and time delay issues and cannot be used to directly track heart valve motion. To solve this problem this group tested several different predictive filters including a Time-Varying

Fourier Series Model with Extended Kalman Filter, an Autoregressive Model with Least Squares Estimator and an Autoregressive Model with Fading Memory Estimator. Each filter was tested via simulation and experimental robot servoing to test their performance given measurement noise, sudden heart rate change and continuously changing heart rates. The experimental robot servoing was performed using a small Phantom 1.5 robot from Sensible Technologies (Woburn, MA) inside a water tank. This study concluded the Extended Kalman Filter (EKF) resulted in the strongest accuracy and robustness overall. Given the conditions tested the EKF carried an RMS error of 1.5mm when tested on clinical heart rate data and 0.97mm for the fixed rate servoing. The results of the 3DUS systems show a very promising method for real time anatomy tracking, but still carry the increased cost associated with ultrasound systems.

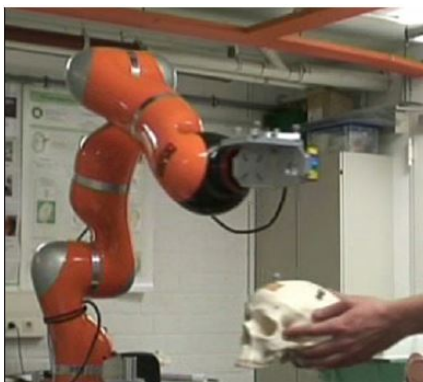


Figure 2.22: Visual Servoing of the KUKA-LWR arm. [59]

More recently Monnich et al. [59] have developed the OP:sense system which combines a seven camera optical tracking system with two KUKA LWR robots to perform accurate automatic positioning tasks (Fig. 2.22). The goal of this research was to combine RMIS techniques with automatic positioning systems in order to work towards semi-autonomous surgical robotics. The KUKA arm can inherently track its own position using the inverse jacobian matrix and has an accuracy of about $30 \mu\text{m}$. The authors then combined this data with the optical tracking system in a registration matrix in order to filter out errors in the individual position tracking systems. This information is then used to implement a control algorithm, which compensates robotic position in relation to respiratory motion. Their iterative learning algorithm repositions the robot

arm each cycle given errors in relation to the respiratory motion, but this convergence is slow and therefore is an area for improvement. The authors also mention incorporating color and depth cameras (RGB-D) such as the Microsoft Kinect © (Microsoft Corporation, Redmond, WA) for human tracking and gestures integration.



Figure 2.23: Basic Light-Stripping setup. [60]

Given the previous work done on the subject of anatomy tracking, it seems the best methodology to pursue would be tracking using a high frame rate camera that ideally can also track depth. One such option would be the Microsoft Kinect. This device utilizes light stripping technology to track depth while simultaneously retrieving an rgb video feed. The light-stripping method works by projecting a known pattern of light onto an object and then using a camera to capture how the pattern varies over a given object (Fig. 2.23). In the case of the Microsoft Kinect, this light pattern is projected in the IR spectrum and therefore is not visible during use. The Kinect currently has a 30 FPS limit, which is low in comparison to some commercially available high resolution CCD cameras. However, for relatively slow moving and static anatomy, a high frame rate will not be crucial.

Other designs could include stereoscopic video setup utilizing high frame-rate, mini CCD cameras such as the Guppy Pro F-031 (Allied Vision Technologies, Stadtroda, Germany). Given the high frame of 124 FPS and modest resolution (656 x 492 pixels), the use of 2 Guppy cameras should provide sufficient data to perform robust stereo matching algorithms. In either setup, an rgb-d camera source will ideally be coupled with a predictive model to track multiple sources of physiological motion.

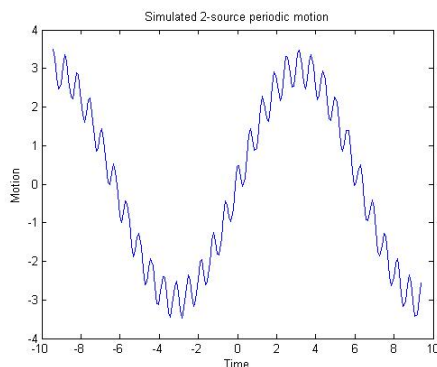


Figure 2.24: Example of a 2-source periodic motion, illustrating a signal with 2 separate contributions.

This predictive model will likely be similar to the adaptive observer method described in [57]. This method incorporates a correction term for the visual measurement and then samples the motion at a high frequency so that the individual respiratory and heartbeat motions can be separated. The individual respiratory and heartbeat motions have distinct frequencies and amplitudes and therefore can be identified and separated (Fig. 2.24). This separation can be done using adaptive filtering which extracts the strongest periodic components via a summation of sinusoidal terms.

Since no system of this kind had been commercialized, it is difficult to speculate on the realistic cost associated with robust anatomical tracking. But as previously mentioned, high quality cameras are a relatively inexpensive component for a tracking system. While not a trivial system, this methodology seems to be the ideal way to track anatomy, especially where it requires motion analysis.

2.4 Gap Analysis and Research Direction

2.4.1 Tool Tracking

The current status of robotic and laparoscopic training research has shown that given the right training tasks, training on either virtual or physical trainers can have a positive correlation to better performance by surgeons in the operating room [61], [62]. In order to accurately gauge proficiency on these training modules several important metrics

need to be quantified, all of which require accurate tool tracking. These include path length, divergence from ideal path length, economy of motion and motion smoothness (Eqs. 2.2). In addition to these standard metrics, research has been done on advanced motion analysis involving hidden markov models (HMM) [63]. The HMM method uses statistical distributions to analyze the motion of a given object.

All of the currently used metrics and the unexplored methods such as HMM and path-to-target all require tool tracking. In order to obtain objective and repeatable measurements of these metrics, the tool tracking must provide accurate spatial information. In order to enable proximal or immediate feedback to boost training effectiveness, it should perform at low latencies and ideally at video frame rates. As the development of standardized testing for RMIS systems progresses, it will be important to develop tracking systems to accompany the modules.

Between the four current system types for tracking of laparoscopic tools (Mechanical, Electromagnetic, Optical and Auditory), no one technology illustrates a clear advantage; each has benefits in either accuracy, robustness or cost. However, in terms of investigating their applicability to robotic systems, it appears that both the mechanical and electromagnetic options are ill-suited to the task of tracking the larger end effectors found on the da Vinci and similar robots. Therefore in addition to the error-prone tracking inherent to the kinematics of the robot arms and compliant tools, these end effectors will very likely require either the optical or auditory options.

Table 2.2: A visual representation of tool tracking technologies and the various requirements they meet.

Method	Real Time	Low Cost	Absolute	Tool Agnostic
Gimbal (Mechanical)	X			X
Ultrasound (Auditory)	X		X	
Magnetic	X		X	
Color Segmentation (Computer Vision)	X	X	X	
Geometric Constraint (Computer Vision) *	X	X	X	X
Physical Model (Computer Vision)		X	X	

*: This geometric constraint approach is not currently available, but is a possible solution.

2.4.2 Anatomy Tracking

Given the relatively unexplored field of robust anatomical tracking, the current options are more limited. The review of the literature uncovered mostly visual servoing and fiber optic sensing had been used for anatomy tracking. The vast majority of these systems were developed primarily to track the frequencies associated with respiratory or heartbeat motion of organs and chest cavities. The visual systems range from texture tracking to colored bead tracking while the fiber optic systems simply provide 1 dimensional depth measurements when aimed at the oscillating tissue in question. It seems that future development using either a light striping RGB-D camera or high-speed CCD cameras with stereoscopic algorithms will provide the best solution for robust, low-cost, and accurate anatomy tracking.

The final synthesis of data acquisition technology would be to combine tool tracking systems with tissue tracking in a single unit. This would keep the already cluttered surgical cavity free from excess devices and allow an easy method for tracking tool position relative to vital organs. This combination could keep surgeons from inducing unnecessary trauma in a patient while using RMIS systems and could also allow surgeons to become accustomed to these systems in a more informative and yet low risk

environment. This sort of information could also result in several new metrics of skill, previously unavailable to researchers which could potentially be more effective in skill discrimination.

2.4.3 Research Direction

It is apparent that both surgical tool and anatomical tracking, if performed with a high level of speed and accuracy could have many benefits to the fields of surgery and medical training. There exist a myriad of systems which have been developed to track and monitor both tools and anatomy separately. For tool tracking there are options which utilize mechanical, ultrasound, video-based and electromagnetic methods. For anatomical tracking, the technologies range from ultrasound to video-based to infrared intensity-based. However, for the purpose of tracking minimally invasive surgical tools and surrounding anatomy inside the human body, it appears many of the systems suffer from various shortcomings. These issues stem from accuracy, condition restraints, cost and robustness among others. A review of the literature revealed that the most promising techniques for tracking surgical tools is either the ultrasound approach or a computer vision based approach. In terms of anatomical tracking, it is apparent that a computer vision based approach is the most likely candidate. The synthesis of a tool tracking method with anatomical tracking would add additional benefits to surgeons and students alike.

The most evident technology for performing a combination of anatomical and tool tracking would be a visual system that could simultaneously track the end effectors and the tissue directly behind it. The computer vision based approach was determined to have clear advantages in both areas of tracking and so is the most compatible solution. There are clearly inherent issues that would accompany such a device, including strong occlusions preventing the camera from recording the entire related anatomy. This would require the development of predictive algorithms to infer the motion of the entire anatomy from only the un-occluded regions. However, such a device first requires both a robust tool and robust anatomy tracking system before such a hybrid could be conceived.

An obvious application of this dual tracking system would be performing basic surgeries on the da Vinci with motion assistance from the computer to maintain a safe

operating space. Applications could also include co-registration of anatomy from MRI indicated target sites, this could be used to identify areas of interest in-vivo as determined from prior analysis.

The development of a complete anatomical tracking algorithm is beyond the scope of this work. Such an algorithm would require an enormous effort in order to compile a library of known representations of all the different human anatomical structures. However, the development of an algorithm capable of tracking the surgical tool is not so arduous. For this reason a tool tracking algorithm will be designed to a high enough performance level so that it can be integrated with an anatomical tracking algorithm in future research. In order to design an algorithm that can be easily incorporated into future anatomical tracking work, the computer vision based approach was elected. This will allow future researchers to approach anatomical tracking with a computer-vision based method and use the work presented here as step to build upon. This work will also provide high enough performance standards so that the methods used for anatomical tracking can incorporate this work without loss of performance.

The algorithm presented for the tracking of surgical tools is designed in such a way as to meet the criteria outlined in Table 2.2. An algorithm capable of real-time, low-cost, absolute and tool-agnostic tracking is immediately applicable in the field of minimally invasive surgery. A system meeting all four of the requirements is not currently available and is therefore the subject of this work.

Chapter 3

Algorithm Design and Selection

3.1 Design Goals

The goal of this research was to design a computer vision algorithm for the automatic localization and tracking of surgical robotic tools in an anatomical environment. A computer vision approach was chosen because of the low cost of materials and widespread development of new, promising techniques. The computer vision field of research has been dated to the 1960's when Marvin Minsky assigned his researcher to use a computer to identify a scene from a television. This field has grown immensely in the last decade due to the widespread proliferation of sophisticated computing systems and novel computer algorithms. A careful review of the literature concerning the visual tracking of surgical robotic tool provided a list of requirements and goals for the development of a new system. This list of goals includes:

- **Speed:** The system should be able to track tools with minimal latency to enable feedback reporting in near real time. Since the bandwidth of surgical tool motions has been experimentally determined as falling below 8Hz [22], we need to achieve at least double that frame rate (16 Hz) to achieve the Nyquist sampling criterion.
- **Accuracy:** The tool should be correctly localized at least 86% of the time that it is within the field of view, this is adopted as a minimal benchmark based on the reported pixel accuracy of Reiter et al [47]. A correct localization will be defined as being within 8 mm of the true position. This benchmark is adopted since it corresponds to the working area of the standard da Vinci tool wrist.
- **Timing Data:** Accurate timing data should be available with each known location for the purposes of computing velocity and acceleration changes.
- **Robust:** The algorithm should maintain accuracy in the face of changing conditions such as light or specular reflectance and should therefore not depend on color segmentation.
- **Tool Agnostic:** The system should not require the use of any specific tool or any platform. Any surgical tool should be tracked by this algorithm regardless of end effector shape, wrist style, or manufacturer.
- **Simplicity:** To maximize its beneficial impact to this research field, the algorithm used should not require extravagant computing power and thus be able to run on a typical laptop. The program should be open source and easily distributable with supporting libraries.
- **Ease of Use:** No additional set-up should be necessary in the operating room, only an existing live video feed or saved stereo video file need be present for the program to properly function.

Figure 3.1: The design goals for the tracking algorithm

The environment chosen for the development of this tracking algorithm was the C++ ISO 11 programming language. The reason for this choice was the availability of

sophisticated open source libraries for computer vision algorithms and other application development. The computer vision library utilized is the Open CV library from Willow Garage [52]. This library includes a large array of functions for capturing video and processing images. In order to include a graphical user interface and OpenGL graphics functionality [49] the Qt library from Digia [64] was also utilized.

3.2 Algorithm Development

In the case of this work, ‘algorithm’ will refer to a synthesis of computer code and mathematics which will be used to achieve a computational goal. The goal of this algorithm will be the detection and localization of a surgical tool tip in any given image (Fig. 3.2).

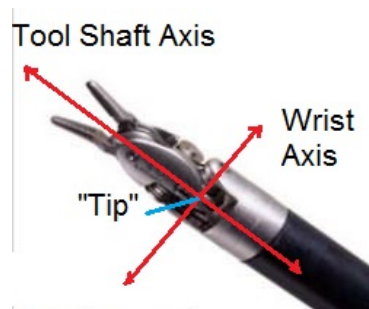


Figure 3.2: A da Vinci tool tip. The ‘tip’ location is indicated by the intersection of the two axes. Source: Intuitive Surgical (Sunnyvale, CA)

The general methodology for developing this tracking algorithm utilized two separate components. The first component is the detection of the 2-dimensional pixel location of the tool tip (Fig. 3.3). For the purposes of this work, the tool tip is defined as the intersection of the axis of the wrist of the end effector and the tool shaft axis (Fig. 3.2). In other words only the end of the tool shaft is considered and not the end effector wrist, this allows the algorithm to be agnostic to tool type. After the tool tip has been localized in pixel space, the second component, stereo vision, is used to extrapolate the 3-dimensional location. Obviously the accuracy with which this location is represented greatly depends on the resolution of the camera feed used. Initially this resolution was set to 640x360 pixels for development and was adopted as the lower bound of acceptable

resolutions.

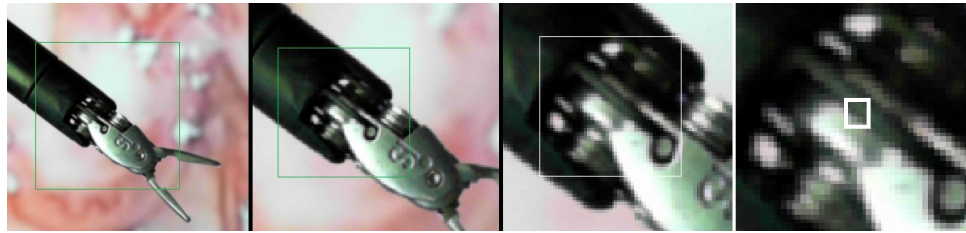


Figure 3.3: Localization of the tool tip in pixel space.

In order to begin testing different algorithms a camera setup was developed using 2 webcams and a windows computer. More information on this setup can be found in section 3.7.



Figure 3.4: Microsoft Lifecam Studio Webcam, the optical axis is indicated in green.

In general, the flow of information for 3-D tool tracking followed a basic outline. This outline is functionally decomposed into 3 major steps: 2-D localization, depth extraction and Cartesian coordinate calculation. The required input is simply a captured video frame and the output is a data log of Cartesian coordinates and time (Fig. 3.5).

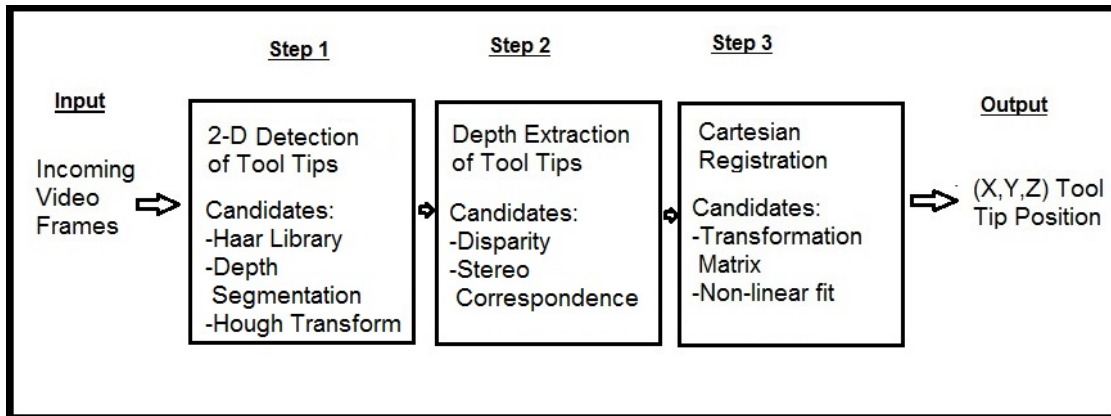


Figure 3.5: The 3 step approach undertaken for object tracking.

Various methods of realizing each step were evaluated. The methods for 2D object detection are presented in detail in section 3.3. These methods are then evaluated in section 3.4. For the depth extraction step, the potential methods are described in section 3.5. For the Cartesian registration, the methods are outlined in section 3.6.

3.3 2D Object Detection Step

The 2-D object detection component of the tracking algorithm was the first component to be developed. In the scope of this work 3 distinct object detection methods were developed. Each method was pursued in enough detail to assess the validity of that particular method. The overview of each method is presented here and analyzed in section 3.4.

3.3.1 Haar Feature Library Approach

The first method pursued for 2-D detection of the surgical tool involved developing a Haar feature based classification library and using that library to identify the tool; similar to the method pursued by Reiter et al. [47]. The OpenCV library contains a built-in function for developing this library via Haar feature classifiers. Haar features for image detection were initially proposed by Viola and Jones [46]. The basic idea behind

Haar features is that instead of analyzing each pixel of an image separately, the image is broken up into tile regions or windows. For each window the sum of pixel intensities in adjacent windows is calculated and the difference is taken between those regions and the target window. This difference is used to classify the target window. When this window is passed over the entire image each small tile can be classified into a certain category. These windows are then compared against known feature classifiers from a library in order to search for the object in question. The feature library is constructed from a large sample of images of the object to be tracked in various configurations and poses. These images are superimposed on a background image not containing the tool and the Haar description for the tool in that configuration is recorded (Fig. 3.6).

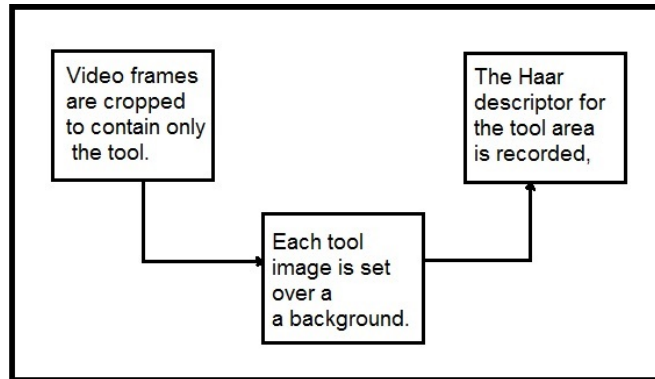


Figure 3.6: Haar training description.

Several thousand frames were captured from surgical video and parsed into small frames featuring the surgical tool (Fig. 3.7). Once each frame was manually cropped to feature only the tool tip region, the sample image group was analyzed with the OpenCV ‘TrainCascade’ program in order to produce a cascade classifier library. This classifier library is compiled into an XML file which can then be queried by the OpenCV functions ‘CascadeClassifier’ and ‘detectMultiScale’. These functions provide a list of all (x,y) pixel locations where the classifier threshold has been found.

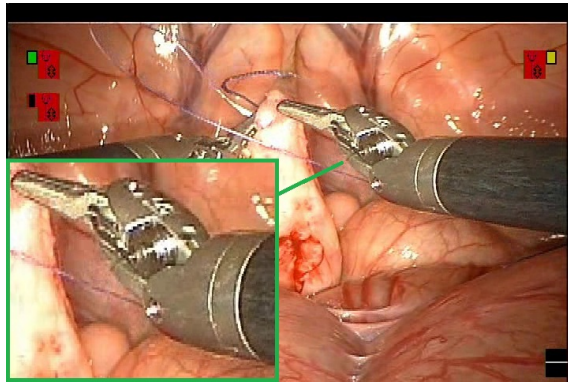


Figure 3.7: Example Haar training frame.

In order to simulate a surgical background for benchmarking and algorithm testing purposes, an image of internal anatomy was extracted from a surgical video and enlarged in order to print the image onto a medium size poster (Fig. 3.8). This poster was then placed below the camera to act as a background for the surgical instrument. While this does not perfectly simulate surgical environment it does provide a rough approximation of the texture variations and colors experienced in a clinical setting.

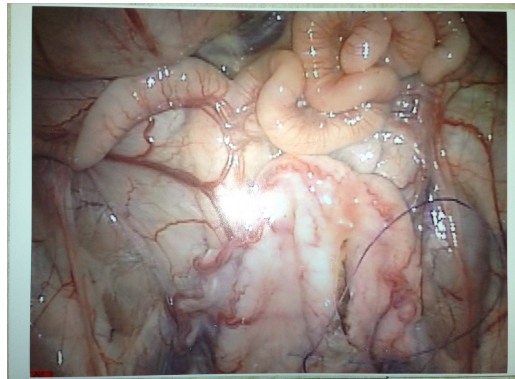


Figure 3.8: Anatomical image used for a background.

Once this cascade classifier library had been compiled, a sample program was written to assess the accuracy and effectiveness of tracking the surgical tool. The sample program used the Lifecam webcams as a video capture source and then extracted each

frame of the video stream as an OpenCV ‘Mat’ container. Each frame Mat was analyzed using the ‘detectMultiScale’ function and each detected instance of the classifier was then drawn as a rectangle on the screen.

3.3.2 Depth Segmentation Approach

The next method that was pursued was the use of stereo depth video and the fact that in a surgical setting the tool shaft is generally closer to the camera than the anatomy. This knowledge can be used to segment the tool pixels from the background. Two cameras in a stereo configuration are necessary to extract depth from a scene in much the same way that humans need two eyes in order to perceive depth (Fig. 3.9). The stereo camera hardware design is explained in greater detail in section 3.7. Stereo vision allows the distance between objects in the right and left source to be compared and used to extrapolate depth.



Figure 3.9: The stereo camera setup. The right and left cameras are labeled, seemingly in reverse since the cameras are pictured from the front. (See section 3.7 for details.)

With the cameras mounted in a stereo configuration, a calibration step was necessary to convert the pixel coordinates in the image into real world (X,Y,Z) coordinates. The key of component of stereo vision is extracting the depth or ‘Z’ component of the coordinate data. The depth component is related to what is referred to as disparity. For any given object visible in both cameras (or channels) field of view, disparity is the difference (in pixels) between the two objects (Fig. 3.10). Once the depth has been calculated from the disparity, the real world X and Y coordinates can be calculated

via direct proportionality to the distance away from the camera. The crucial element in determining the disparity and depth of an object is being able to correctly identify common features in both channels of a stereo video feed.

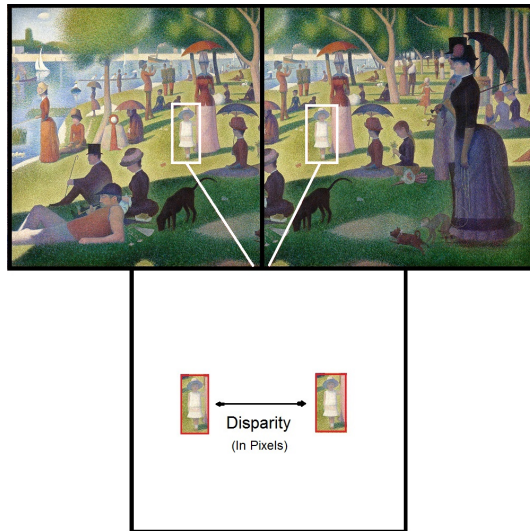


Figure 3.10: Disparity is the difference in pixel location between the same object visible in two cameras.

For the purpose of this tracking algorithm the global (X_w, Y_w, Z_w) world coordinate system has been defined so as to make the mapping from pixel coordinates to world coordinate as intuitive as possible. To this end the world Z_w axis has been defined as extending straight out from the middle point between the two cameras lenses with the origin at the plane formed by the camera lenses (Fig. 3.11). The world X_w axis has been defined as being parallel to the pixel x_p axis, i.e. horizontal across the image plane with the origin placed at the middle point between the two cameras. World X_w thus increases to the right in the image plane. Finally the world Y_w axis has also been defined to be parallel with the pixel y_p coordinates and thus oriented vertically in the image plane with increasing values going up in the image plane. The origin of the world Y_w is defined to be coincident with the X_w and Z_w origins at the midpoint of the two cameras. While this coordinate orientation does not comply with the standard right hand rule of coordinate systems, it makes the transference from pixel to global coordinates more intuitive. In addition to this the pixel coordinate system is defined to have an origin at

the center of the image with pixel x_p coordinates increasing to the right and pixel y_p coordinates increasing in the up direction (Fig. 3.12).

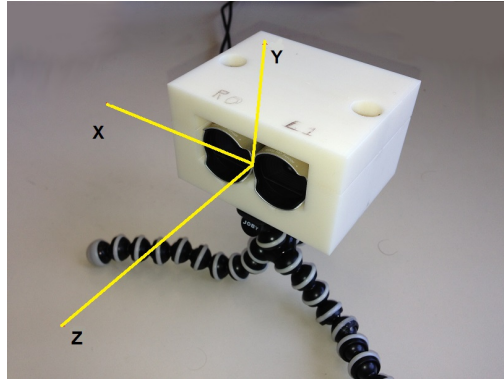


Figure 3.11: The 3D camera coordinate system is portrayed in red.

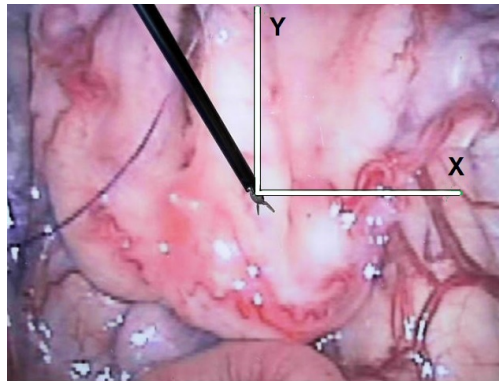


Figure 3.12: The 2D pixel coordinate system is portrayed in green, Z points into the page.

The coordinate system must also acknowledge that for any given object visible in both channels, the pixel location of that object will have different (x,y) locations relative to each images coordinate frame. While this is crucial for determining disparity and depth, it also yields multiple options for determining location dominance. For instance, most humans are right eye dominant meaning they gain most of their visual input from their right eye. They then use their left eye mainly to infer depth. In the stereo camera setup we could use either the right or left pixel locations alone and use the other channels

information only for depth. However, in order to avoid using right or left dominance, this work will use the average of x and y locations as the true pixel coordinates. In the stereo setup this translates to:

$$y_{avg} = y_{right} = y_{left} \quad (3.1)$$

$$x_{avg} = \frac{x_{right} + x_{left}}{2} \quad (3.2)$$

This is due to the fact that the stereo cameras are aligned in the ‘xz’ plane but separated in the ‘yz’ plane.

Instead of manually identifying common objects in both stereo channels and comparing their pixel positions, it is often easier to use basic feature extraction and calculate the disparity for all strong feature candidates. In the case of OpenCV this feature extraction is implemented using the Block matching stereo correspondence technique. Block matching works by finding strong patterns of pixels in each channel of the stereo camera and compares the pixel difference along the ‘epipolar line’. The epipolar line is defined as the horizontal line along the same row of pixels in an image. In order to correctly implement this algorithm, the stereo camera setup needs to first be calibrated in order to rectify any offset in the orientation of the cameras and determine the transformation between pixel and global space. The OpenCV library provides functionality for performing stereo calibration which is based on the work of Zhang [65]. The method for calibrating the stereo cameras requires a chessboard pattern to be printed on a sheet of paper (Fig. 3.13). The chessboard pattern is then observed from multiple angles by both cameras in such a way that each internal corner on the chessboard can be identified in each frame (Fig. 3.14).

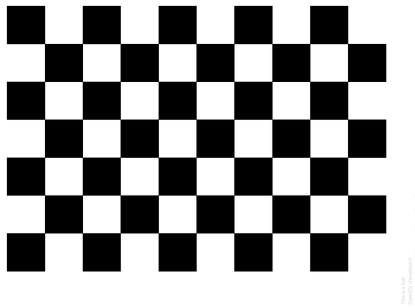


Figure 3.13: OpenCV Calibration Chessboard.

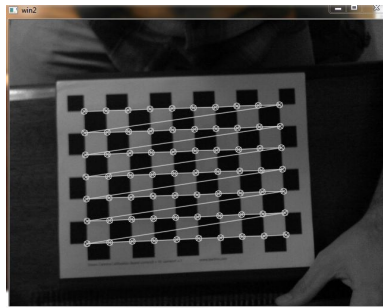


Figure 3.14: OpenCV Chessboard with corners identified.

The corners on the chessboard are then enumerated and the disparities and offsets are recorded for each corner and used to calibrate intrinsic and extrinsic camera values. This calibration data can then be used via the OpenCV ‘StereoRectify’ function in order to correctly align each stereo channel before performing the block matching correspondence algorithm. The block matching correspondence algorithm simply identifies key features in both images and then determines the disparity between matching features in both channels.

The stereo depth calibration was implemented in a test program and used to manually inspect the effectiveness of the depth extraction algorithm. A live video stream from both of the stereo cameras was read into an OpenCV Mat container and rectified using the ‘Remap’ function. Using the rectified images the two channels were then used to compute depth using the ‘BMState’ function operator. The depth calculations were then scaled so as to fit into a 0-255 single channel grayscale image.

3.3.3 Hough Transform and Geometric Approach

The geometric constraint approach pursued in [45], [42], [43], while relatively simple seems to maintain a higher level of robustness than simple color tracking while obtaining a faster computational speed than Haar tracking or similar approaches. However, several of these prior approaches relied on physical constraints requiring additional measurements which were not appealing. Therefore a third approach was identified in which the known geometric constraints of the tool shaft could be exploited using a Hough transform. For any given RMIS or MIS tool, the known geometric constraints include the following:

- The tool shaft is long (relative to other lines in the image).
- The tool shaft is straight.
- The tool shaft originates from the sides of the field of view.
- The end of the tool shaft in the middle of the field of view corresponds to the tool tip.

Figure 3.15: Geometric constraints for a tool shaft based on observation.

This approach does not require any a priori measurements but simply scans the image for two parallel lines of sufficient length, and with ends that are nearby a previously identified location. The general idea behind this concept is depicted in Fig. 3.16. After finding the primary edges within the image the, the most prominent lines are then found using a Hough transform.

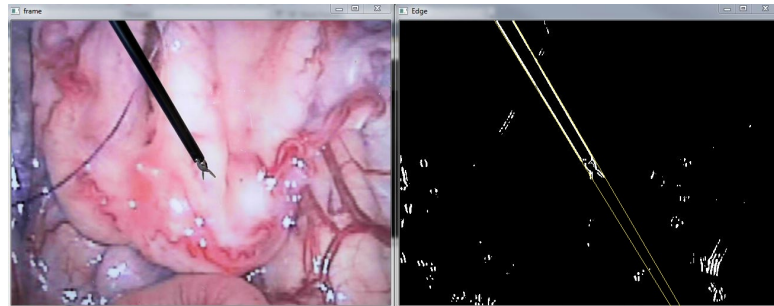


Figure 3.16: Tool shaft with edge detection and lines.

The first component of this geometric approach is the isolation of the edge along the tool shaft, otherwise known as edge detection. First the image must be blurred in order to only detect prominent lines and mitigate noise. The blurring effect is achieved by passing a normalized box filter over the image. A normalized box filter takes the form of a 3x3 kernel (K_{nb}).

$$K_{nb} = \frac{1}{9} \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \quad (3.3)$$

Edge detection is the process of analyzing pixel intensity variations in order to identify areas of large jumps in pixel intensity. The magnitude with which intensity changes for a set of pixels is known as the gradient and therefore edges can be quantified as pixel regions with large or ‘strong’ gradients (Fig. 3.17). On the other hand pixel regions with small or ‘weak’ gradients typically do not represent edges (Fig. 3.18). Once the gradients for all pixels have been calculated, the edges can be picked out by setting a threshold value for the magnitude of the gradient. Higher thresholds mean only stronger edges will be represented. Edge pixels are typically indicated by white pixels while black pixels represent non-edge areas.

Pixel Value	200	200	200	190	10	10	10
Gradient							

Figure 3.17: A 1D representation of strong pixel gradients.

Pixel Value	200	180	150	75	75	10	10
Gradient							

Figure 3.18: A 1D representation of weak pixel gradients.

While there exists a plethora of algorithms which perform edge detection, the two most common algorithms are the Sobel and the Canny edge detector. In the case of the Sobel algorithm the edges are detected by passing a kernel or matrix over each pixel in the image, a process known as convolution. For a kernel of any given size ($n \times n$), the image is convolved by taking each ($n \times n$) sub-matrix 'A' from the image and multiplying each pixel element wise with the corresponding element in the kernel. Such a calculation approximates the effect of taking a 2 dimensional derivative across the image. This kernel takes the form of S_x and S_y below. The actual gradient is then calculated by taking the magnitude of the each kernel convolution (Eq: 3.4).

$$S = \sqrt{(S_x * A)^2 + (S_y * A)^2} \quad (3.4)$$

Where A is the sub-matrix from the image
and S_x and S_y are defined as:

$$S_x = \begin{vmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{vmatrix}.$$

$$S_y = \begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}.$$

Note: here the * operator symbolizes the convolution operator

In the case of the Canny edge detector, the algorithm functions by taking a first derivative across the image, similar to the Sobel version. However this derivative is taken in multiple directions depending on the connectivity desired and also returns the ‘angle’ of the gradient. This approach delivers edges with sharper angles than the Sobel approach. Each directional derivative is used to calculate the gradient using a non-maxima suppression based on the gradient angle calculated. The non-maxima suppression rounds the angle of each gradient to the nearest 0 deg, 45 deg, 90 deg, or 135 deg. For any given rounded angle, the neighboring pixels along that same angle are analyzed to determine if a particular gradient is a local maximum or not (relative to it’s neighboring pixels). The Canny method then also incorporates a hysteresis thresholding step after the gradients have been calculated. The hysteresis thresholding involves a second pass over the image to assign votes based on a low and high gradient threshold. Any gradient over the high threshold is certainly an edge, any gradient below the low threshold is certainly not. However any gradient which lies between the high and low threshold is then only considered an edge if it lies next to another certain edge. Such a system allows edges to be filled in the case of blurriness or occlusions. While the Canny operator does perform better in terms of robustness, this method also has a slightly higher computational time than the Sobel operator. For this reason both methods were tested in order to determine superiority. As can be seen in Fig. 3.18, both methods accurately identify the edges on either side of the tool shaft and in some instances the Sobel operator returns a thicker edge, which is beneficial in the case of Hough transforms since those lines will be more pronounced.

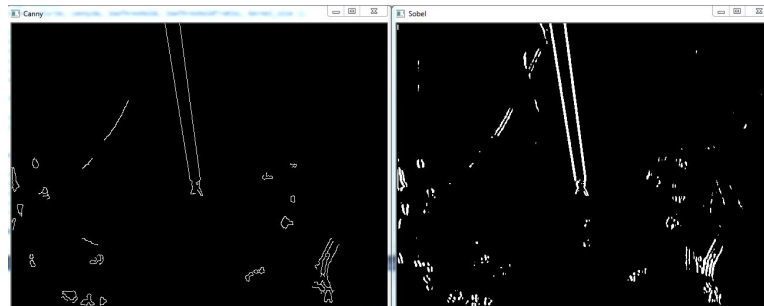


Figure 3.19: Canny(left) and Sobel(right) edge detection.

Once a successful edge detection had been implemented, the edge image was subjected to a Hough [“hoff”] transform algorithm in order to detect and classify the lines present in the image. As discussed the Hough transform was patented in 1962 as a method for detecting geometric patterns in images [44]. The Hough algorithm uses a two dimensional voting array in order to cast votes for lines characterized by ρ and θ . These two parameters can be used to describe any line in a 2D plane using equation 3.5.

$$Y = -\frac{\cos(\theta)}{\sin(\theta)} * X + \frac{\rho}{\sin(\theta)} \quad (3.5)$$

For any given point, an infinite number of lines cross through that point. However for any two points only one line goes through both (Fig. 3.20). In the Hough transform, each white pixel (edge pixel) contributes 1 vote to each line that passes through it. Each vote is distributed into bins in a 2D array corresponding to the ρ , θ parameters that describe. For example, an image with many points all on the same line and no points anywhere else, the number of votes in the ρ , θ bin corresponding to that line grows for each point lying on that line. In this example all the other bins would have only 1 vote in them. Once the voting process is complete, a threshold is used to determine the bins with sufficient votes and those ρ and θ values. Once the bins with the highest votes have been found those values can be used to superimpose the lines over the image.

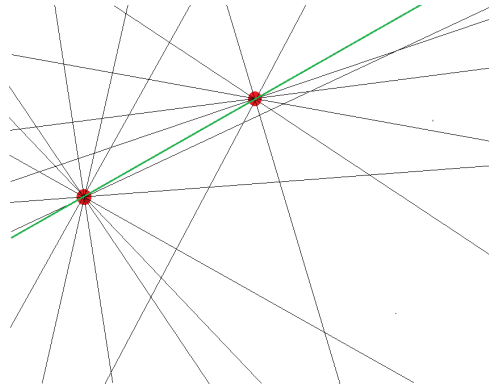


Figure 3.20: Lines intersecting 2 points, only 1 line (green) intersects both.

The traditional voting method for the Hough transform can become quite time consuming given a large image resolution or a large number of white (edge pixels). In order to deal with this issue, Matas et al. [66] developed an augmented Hough transform; the ‘Probabilistic Hough Transform’ (PHT). The PHT improves computation time by not considering all the edge pixels provided. Instead this algorithm uses only a subset of input points based on the specific complexity of the particular image provided. Images with more complex and varied edge pixel patterns will inherently require more sample points while more elementary patterns will require less points. The use of this probabilistic subset of points has been shown by Matas et al. to demonstrate negligible drop in performance when compared with the standard Hough transform. The OpenCV implementation of the PHT has an additional benefit that instead of returning all the ρ , θ pairs for lines in the image, it returns the (X,Y) endpoint coordinates of each line. The exact end point coordinates of each line allows our algorithm to forgo color segmentation or other similar shaft-tip segmentation efforts since the tip will simply correspond to the line end closest to our physical constraints. In practice this line end will be the end closer to the center of the frame, i.e. farther from the sides of the frame.

In order to examine the efficacy of the PHT algorithm in OpenCV a sample program was written wherein a live camera feed is captured and each individual frame is saved in the Mat container. The frames are first converted to a grayscale image and blurred. The blurring effect is used so that background edges are less pronounced while the black tool shaft edges will still be prominent. Once the blurring step has finished, the edges can

be detected. Each frame is subjected to either a Canny or Sobel edge detection routine followed by the PHT algorithm. Each endpoint returned by the PHT algorithm is then used to superimpose a line on the original frame mat in order to manually inspect the location of lines and their end points.

3.4 Algorithm Evaluation and Selection for Object Detection

Each of the 2-D object detection algorithms was given an initial implementation as described in section 3.2. The initial implementations were used to select an object detection method with sufficient accuracy and robustness. The best method was then identified as the approach that should be used for further development. Using the test Haar feature program, results from the Haar approach were analyzed manually and indicated that the Haar feature method resulted in a large amount of false positives and inaccuracies (Fig. 3.21).

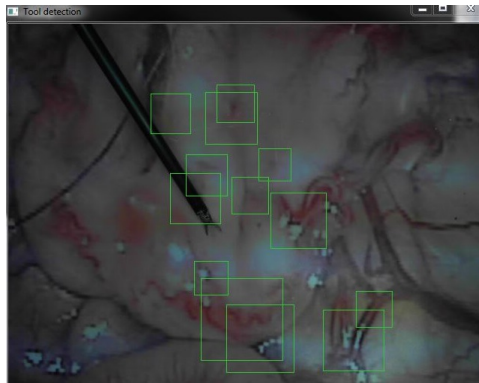


Figure 3.21: Sample tracking frame using the Haar feature classifier.

By adjusting the target window parameter in the ‘CascadeClassifier’ function the number of false positives could be reduced but then the tool itself was often lost (Fig. 3.22). Several other parameters were also adjusted for this initial cascade classifier attempt but without a successful instance of tracking where only the tool end effector was tracked consistently.

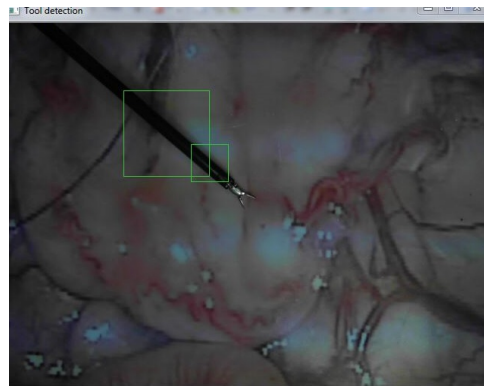


Figure 3.22: Sample Haar tracking with adjusted parameters.

After this initial unsuccessful attempt at using the Haar method, it was assumed that a contributing factor for the false positive and inaccurate tracking was the insufficient sample size of the image collection. In order to remedy this, a longer surgical video was again parsed into more individual frames and cropped in order to feature the tool. The larger amount of sample frames also had the added benefit of depicting the end effector in more and varied positions. This larger sample set was again analyzed with the 'TrainCascade' function and built into an XML classifier library. Using this revised Haar library the test program was again run to manually inspect the effectiveness of the Haar tracking method. The revised library did not function any better than the initial attempt and at times performed worse (Fig. 3.23).

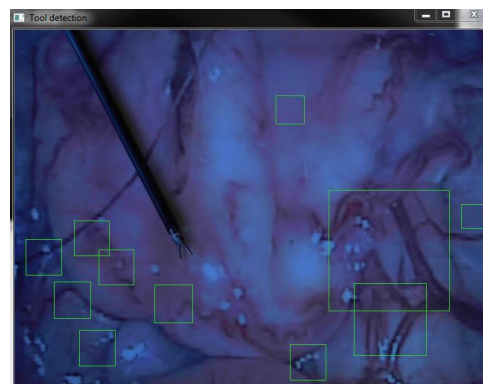


Figure 3.23: Sample Haar tracking with revised library.

Given the lack of success even with a more extensive library of Haar feature classifiers, an extensive analysis of the Haar classifier library was performed to identify the issue with using this method. The primary use for the Haar classifier libraries has been face detection and detection of other large objects [67]. The human face provides medium sized regions of distinct intensity value variations. For example the face will always have two regions of darkness corresponding to the eyes surrounded by lighter regions in the cheeks and forehead. These regions are large enough to provide reasonably obvious pixel intensity sums. Unfortunately the tip of the surgical tool is much smaller and provides only two distinct regions of pixel variation; the end of the shaft and the gray wrist (Fig. 3.24). The major downfall of having only two distinct regions is there exists a plethora of dark regions next to light regions in any image with a textured background. As can be seen in Fig. 3.23, several false positives are simply regions of dark contrasted against light.

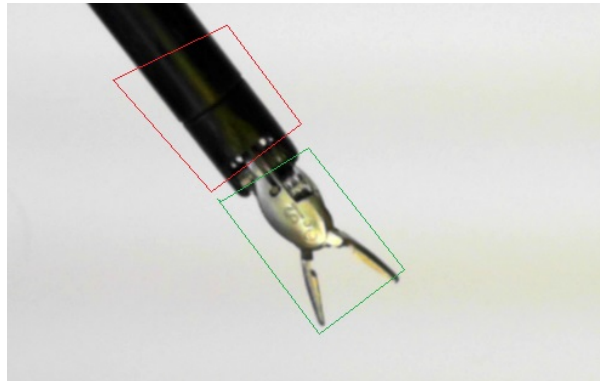


Figure 3.24: The two primary intensity regions of the tool tip.

The Haar classifier approach to object detection was unsuccessful, even with multiple attempts at training the classifier library. The primary issue with the Haar approach was the lack of distinct pixel regions around the tool tip.

Initial indications from the stereo correspondence depth images were promising. As seen in Fig. 3.25 the tool is clearly segmented by color in the grayscale image.

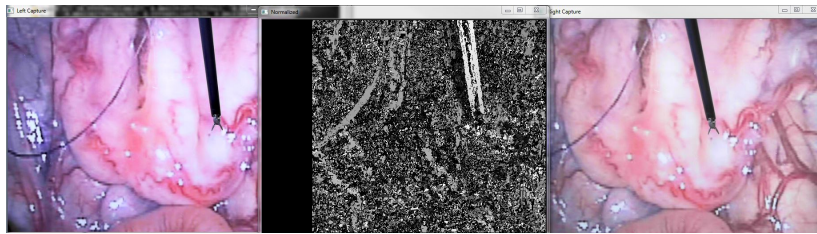


Figure 3.25: Depth extraction using stereo block matching.

However using the grayscale representation of depth, an attempt was made to threshold depth values so as only to extract the tool pixels since they should theoretically be closer than the simulated background image. As can be seen in Fig. 3.26 the thresholded image is very noisy. This noise resulted in a very difficult process of choosing an ideal threshold value. While the noise could have been resolved using a series of low pass filters, the noise also had an unfortunate effect of cutting off the end of the tool tip as the tip approached the background image (Fig. 3.27). The tool tip (along with other key features) were also often lost in conditions involving low light or low texture presence.

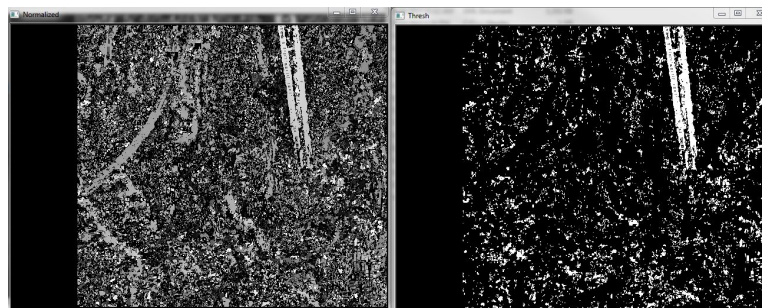


Figure 3.26: Depth threshold using stereo block matching.

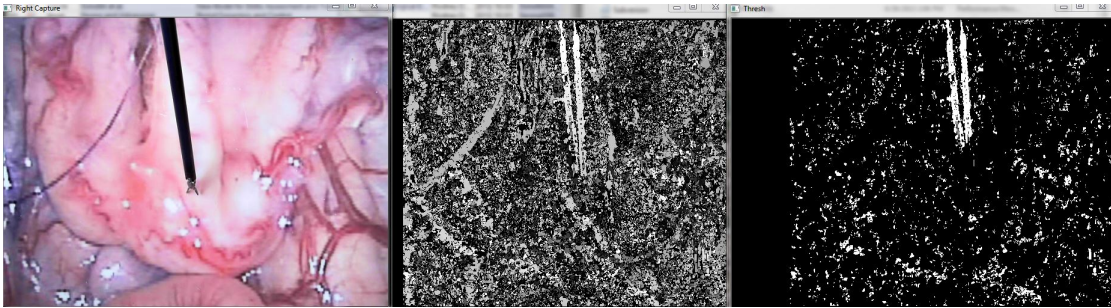


Figure 3.27: Depth threshold with cut off tip.

It is interesting to note that the cost of performing the full stereo correspondence is computationally very high. Stereo correspondence requires on average 52 ms for each pair of stereo frames. The algorithm using the stereo correspondence method has a relatively poor frame rate.

After reviewing the performance and symptoms of the stereo correspondence thresholding method, it was implied that there was too much noise and inconsistency in performance to pursue this approach further.

In the case of the Hough transform - geometric constraint approach, several promising results were discovered. In the sample program the PHT algorithm returns lines which are then superimposed on the original frame mat in order to manually inspect the location of lines and their end points (Fig. 3.28). As can be seen in the figure, only long and consistent lines which appear after the PHT algorithm are those corresponding to the two sides of the tool shaft.

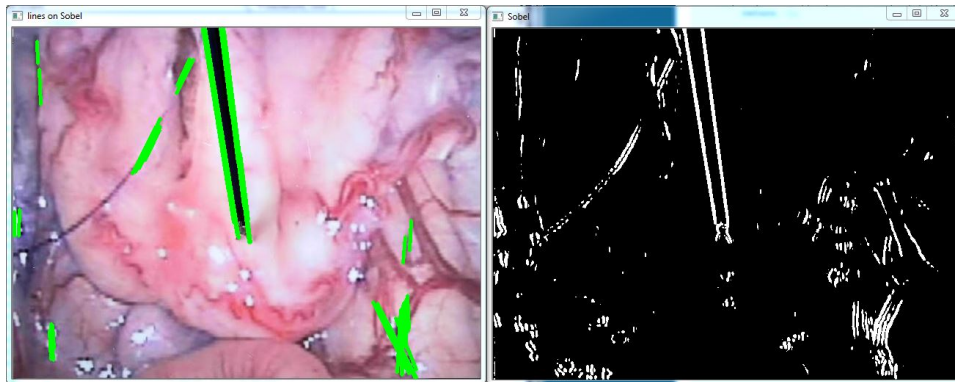


Figure 3.28: Hough lines representing the tool superimposed in green on the image on right.

The hough approach was applied to the tool shaft in several different configurations and locations around the field of view (Figs. 3.29, 3.30, 3.31). The Sobel edge detection was used in this configuration because it provided thicker and clearer lines than the Canny method. Each test configuration provided distinct lines on either side of the tool shaft with relatively few lines found in the background image.

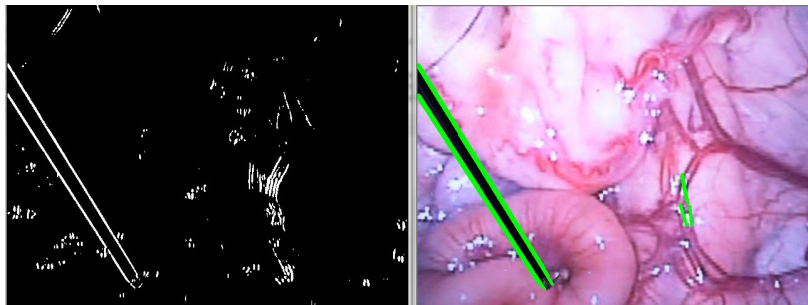


Figure 3.29: Tool configuration 2 with hough lines traced in green.

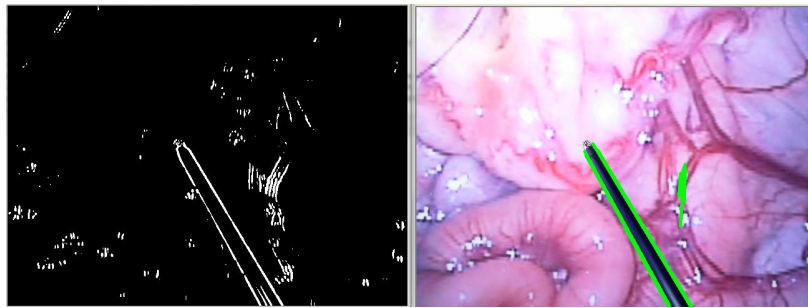


Figure 3.30: Tool configuration 3 with hough lines traced in green.

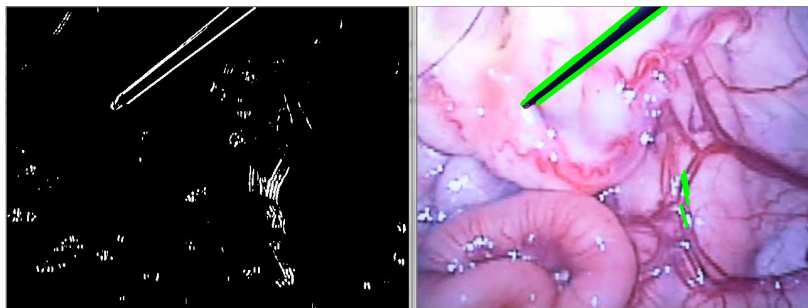


Figure 3.31: Tool configuration 4 with hough lines traced in green.

The results from initial inspection of the hough lines approach to object detection proved to be promising. The bounding lines on either side of the tool shaft were detected in a wide variety of tool configurations. With these lines reliably detected, the geometric constraints outlined in figure 3.3.3 should theoretically be easily manipulated.

Each 2-D object detection method was reviewed for efficacy in reliably identifying the tool location in a given frame. The shortcomings of the Haar library approach stemmed from the lack of large and robust features found on the tool tip. This was visible in the large amount of false positives for tool tracking in the sample program. In the case of the depth thresholding approach to tool tracking, the primary downfall came from the features used to compute disparity. As the tool shaft approached background textures, the tool shaft was often lost or the end of the shaft was cut off. The loss of tool shaft detection in these conditions made the depth segmentation approach an unreliable candidate for tool tracking. The Hough transform approach provided the most reliable detection of the tool shaft. The lines outlining the tool shaft were consistently detected

and therefore available for selection using the geometric constraints found in figure 3.3.3. The success of the PHT algorithm in detecting lines on the sides of the tool shaft and the effectiveness of the Sobel operator in finding edges for this purpose, made the joint Hough Transform-Geometric constraint approach a logical choice for further development.

3.5 Depth Extraction Step

The second component of the tracking algorithm is the calculation of the depth using the stereo camera setup. The method selected for depth extraction is inherently dependent on the method used for tool detection. As described in section 3.3.2, the full stereo correspondence method is the most straightforward method for calculating depth for the whole scene. The works of [42], [45] and [47] all utilize full stereo correspondence method. This method is the obvious choice for the depth segmentation approach, however for the other object detection methods, depth extraction can be achieved in other ways.

The alternative method for extracting depth is to only use the pixel location of the tool tip in both channels. Using the right and left pixel location the disparity can easily be calculated using the distance formula (Fig. 3.32). This method of single tool tip disparity calculation requires far simpler calculation than the full stereo correspondence. The tool tip disparity can be computed using basic algebra and this disparity can then be used to determine the depth to the tool tip.

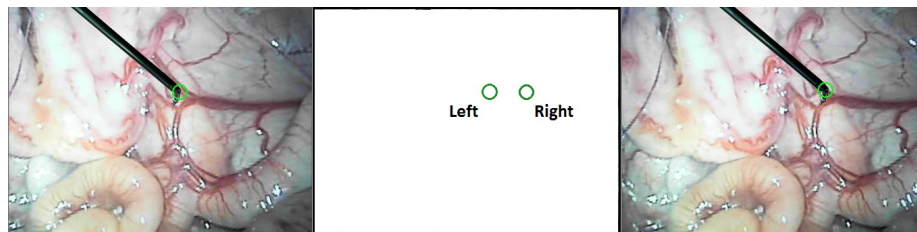


Figure 3.32: The tool is localized in separate channels and then the disparity is computed.

The formula used to calculate depth from disparity depends greatly on the camera

setup and model selected. Several models exist for computing depth. The most ubiquitous of these models is the standard linear transformation outlined in the works of Zhang [65] and Tsai [68]. This linear model uses the form of a 3x3 rotation matrix and a transformation matrix.

The alternative model for the depth calculation utilizes a planar calibration board and collect data correlating depth with disparity. The data collected can be used to devise a model equation with the best fit. This model is likely a non-linear given the variations in camera lenses.

Since the only depth information required by the tool tracking algorithm is the tool tip depth, it makes little sense to compute the entire stereo correspondence depth map. Therefore, the chosen method for calculating tool tip depth will be the single disparity-depth calculation. This method will require detecting the tool tip in both channels separately and then combining the information.

3.6 Cartesian Registration Step

The final step in the tracking algorithm is two correlate the pixel space coordinates of the tool tip with real world coordinates. As indicated in section 3.5, the method used in subsequent tracking steps is dependent on the previous components. All of the depth methods outlined provide the Z component of the Cartesian coordinates. The remaining coordinates are (X_w, Y_w) . The relationship between pixel location (x_p, y_p) and the world coordinates is highly dependent on depth Z_w . A slight movement in pixel location for far away depths results in a much larger movement in global position.

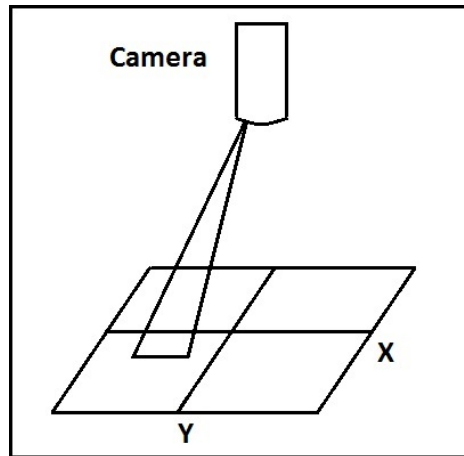


Figure 3.33: Movement in pixel space is amplified in global space for longer depths.

Using this observation it is evident that there should be a direct relationship between depth and global coordinates in form:

$$Z_w \propto \frac{(X_w, Y_w)}{(x_p, y_p)} \quad (3.6)$$

The equation coefficients for this model will be determined after a calibration has been completed. The Z_w calibration as well as the X and Y will be dependent on the camera setup in question. Separate calibrations are required for the experimental camera setup and the da Vinci Endoscope. Independent of the setup used, the model for Cartesian registration will rely heavily on the data relationships observed. This method of Cartesian registration was the only method considered during development. The simplicity and explicit correlation to data make this the logical choice.

3.7 Hardware Design and Implementation

Before testing the tracking algorithm using a da Vinci endoscopic camera, an experimental camera design was required in order to develop the tracking software. In order to build and design this system quickly, two USB webcams were used given the speed with which they can be setup and the portability of the webcams. The cameras used were Microsoft Lifecam Studio webcams. These cameras offer high frame rates and several resolutions (including HD settings). While form factor was not of particular

concern, having a smaller camera setup made it easier to design a stereo mount. The Lifecams can achieve frame rates of 30 FPS at standard resolutions with slight drops at higher resolutions. The available resolutions are 160x120, 320x240, 640x360, 800x448 and 1280x720 (HD). At the time of purchase, each Lifecam cost approximately \$50 USD.

In order to implement a vision system capable of extracting depth, two cameras must be situated side by side with their optical axis parallel. The cameras must be at the same height and lateral position. These two parallel cameras can then extract depth using the disparity in pixel location of given object. The two cameras each contribute images of the same field of view from slightly different perspective, the left and right, thus the term stereo vision. An important consideration in designing a stereo vision is the physical separation between the cameras optical lenses; the interocular distance. Cameras with larger separations are better suited to far away depth fields while very close cameras work better for close up views. For this application several separations were tested for efficacy before a distance was chosen. Initially a separation of 25 mm was used as a test case. However, this interocular distance provided poor performance for depth fields of 200 mm or closer. Slowly the cameras were adjusted to be closer and closer using Lego building blocks. Eventually the camera housings were placed right next to each other which offered the best possible stereo correspondence for nearby depths and so a resulting interocular distance of 29.1 mm was elected.

In order to design a more permanent mount, a rapid prototype mount was designed using computer aided design (CAD). The mount was designed in the PTC Creo CAD environment (PTC, Lansing, MI). The primary purpose of the mount is to hold the two cameras tightly together without allowing rotation or translation of the camera housing. In order to constrain the position and orientation of the cameras, the mount took advantage of certain physical elements of the Lifecam housing. These elements are the USB cable housing (1), the top microphone (2) and the lens guard(3) (Fig. 3.34). The USB housing is used to prevent rotation of the cameras. The top microphone is used to prevent lateral slipping and the lens guard is used to securely hold the camera from the sides.



Figure 3.34: USB cable housing (1), top microphone(2), and lenses guard (3).

The camera mount to hold these two cameras uses a clamping method to hold the two cameras in place. The camera mount consists of a top and bottom case which are held together by a nylon set screw. Each part was designed specifically to match the geometry of the Lifecam cameras in order to hold the two cameras perfectly parallel and at the exact same height and angle.

The top half of the camera mount is designed symmetrically with beveled angles at the front and back of the mount. These beveled angles use the top microphone to keep the camera in place (Fig. 3.35). These angled supports also help force the cameras to sit tightly next to one another. The top half also has through wholes on the sides for the nylon set screws.

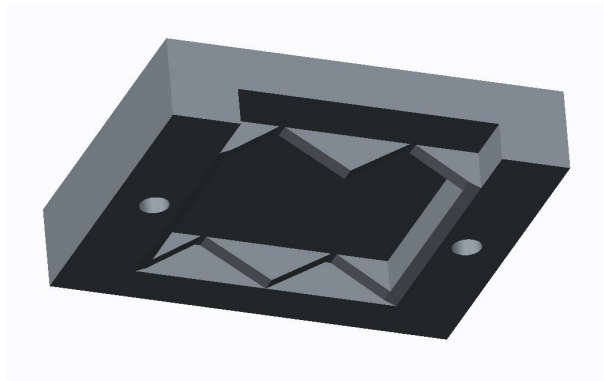


Figure 3.35: The top of the camera mount.

The lower half of the camera mount is also designed symmetrically but with added support holes. The lower mount contains two symmetric through holes which hold the USB cable housing in place, helping to prevent rotation (Fig. 3.36). The lower half also has similar beveled angle supports for added security. Finally, the lower half has through holes which align with the top half so that the set screws can go through.

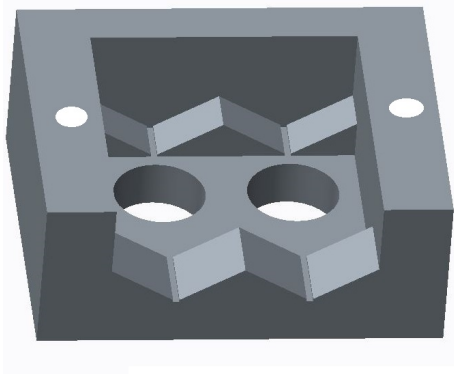


Figure 3.36: The lower half of the camera mount.

The overall camera mount securely holds the two Lifecams together and at a perfectly parallel angle (Fig. 3.37). The design also easily allows clamps and tripods to be utilized for different benchmark and algorithm testing. The design allows easy access to all necessary cables and buttons as well.

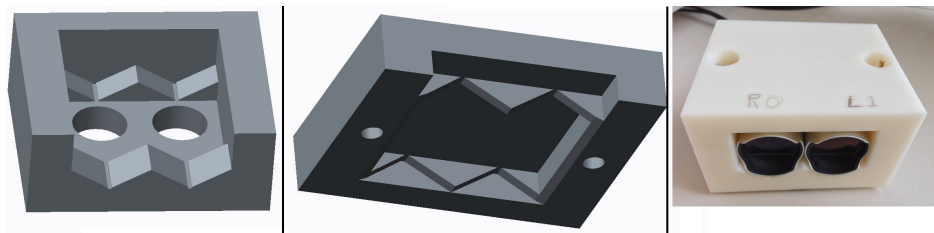


Figure 3.37: The camera mount housing for two webcams.

In order to test the tracking algorithm using the da Vinci Endoscopic camera system, a method for extracting the video frames from the da Vinci cameras was required. In order to accomplish this a Digital Video Interface (DVI) frame grabber was utilized. The frame grabber chosen was the VC200xUSB dual channel DVI box from Electronic

Modular Solutions (Wigston, England) (Fig. 3.38). This DVI frame grabber allows simultaneous frame grabbing from two separate DVI sources. This device also allows the video frames to be synced and read in the computer as a Direct Show filter, allowing seamless integration with existing software including OpenCV. The capture device supports up to 100 frames per second capture rate and simultaneous resolutions up to 1080p. At maximum resolution (1920x1200) the frame rate is limited to 59 FPS.



Figure 3.38: The VC200xUSB frame grabber.

Using the frame grabber, the video signal from both channels of the endoscopic camera (Fig. 3.39) can be diverted into the USB capture box via SDI cables. The SDI outputs are found on the camera control unit of the da Vinci video cart. The SDI signals are then converted to DVI. From the USB connection, both video channels are read by the computer for real time analysis. This video capture device was used for both calibration work and performance analysis for the da Vinci camera tracking. At the time of publication the cost for the frame grabber device was \$ 1000 USD. However, this device is not considered a core component of the tracking technology.



Figure 3.39: The da Vinci camera unit and Endoscope.

The da Vinci Endoscope provides high quality stereo images in order to supply the surgeon with 3D visual information via the master console. The technical specifications for the Endoscope are not made available but were roughly determined using the capture device. The endoscope is capable of resolutions up to 1920x1080 and provides a framerate up to 100 FPS. This setup is relatively similar to the experimental camera setup except for a change in interocular separation. For the Endoscope cameras the separation has been measured to be 5.1 mm.

For computer benchmarking purposes, a Windows computer running Windows 7 was utilized. This PC used an Intel © Core i7 processor running at 3.0 GHz. This PC also possessed 32 GB of RAM. This computer supported the two video sources via dual USB 3.0 ports.

Chapter 4

Implementation of the Complete Tracking Algorithm

The tool tracking algorithm steps, as outlined in section 3.2, consist of the 3 main steps: 2D object detection, depth extraction and Cartesian registration. Having implemented the 3 separate 2d object detection methods outlined in section 3.3, the method were analyzed for robustness and efficacy. In addition, the methods for depth extraction through stereo and Cartesian registration were all presented.

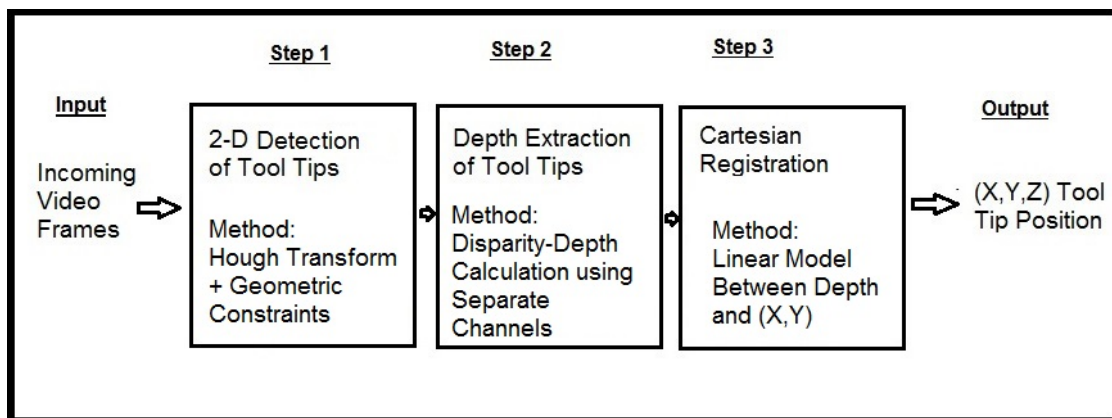


Figure 4.1: The tool tracking approach with the selected methods for each of the 3 steps.

Having analyzed the proposed methods, a single method for each step was selected for further development. The method that was selected as the most efficient in detecting the surgical tool shaft was the joint Hough transform-geometric constraint approach. The method selected for depth extraction was to detect the tools separately in each channel and use the disparity to compute the depth. Finally, for the Cartesian registration, the method selected was a linear model correlating depth and pixel location into Cartesian (X_w, Y_w) . In order to provide the details of this approach to tool tracking, the algorithm implementation is presented in section 4.1. Using this approach, the software implementation is presented in section 4.2.

4.1 Complete Algorithm Overview: Joint Hough Transform-Geometric Approach

Given the success of the Hough transform approach, a more sophisticated program was developed. This program's purpose was to determine the accuracy with which the tool tip could be localized in 3-D coordinates by using the Hough transform detection method. At first this program was written to track only 1 visible tool at a time, with obvious extensions to handle 2 or more visible tools. This program uses 4 main steps to locate the tool; frame capture, edge detection, hough transform, and data sorting. In order to utilize the information provided by the tool shaft edge lines, a list of known geometric constraints pertaining to the shaft lines was compiled. These were selected so as to be generalizable to any tool.

- The lines on either side of the shaft should be approximately parallel, therefore the θ values for each line should be close.
- The endpoints of the two lines should both be a short distance away from each other and the distance between endpoints at the start and end of the shaft should be similar.
- The length of the two side lines should be longer than any other set of parallel lines with a similar θ value.
- For any given pair of lines meeting the aforementioned criteria, there should be no second pair of lines with similar endpoint locations, so as to prevent duplicates.

Figure 4.2: Known geometric constraints for any typical surgical tool (Geometric Priors).

For any pair of suitable lines forming the border of the tool shaft, the PHT algorithm will return 4 endpoints (2 for each line). These endpoints must be sorted in order to find the two points representing the tool tip. To do this, either the points nearest the center of the image or a known previous location are used to isolate the tool tip endpoints and not the endpoints representing the shaft entrance into the image (Fig. 4.3). Once the two desired endpoints have been identified, the midpoint of the two closest endpoints is taken as the tool tip location, since that point should ideally represent the start of the tool wrist or the end of the tool shaft.

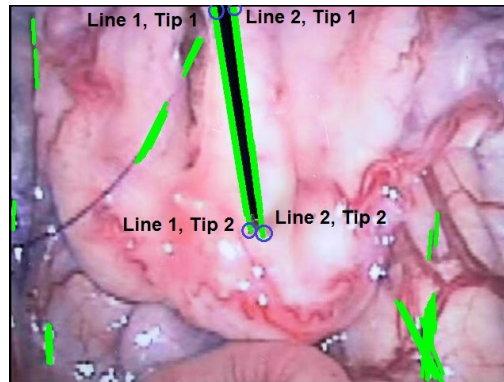


Figure 4.3: The four end points of the tool shaft lines are circled in blue.

With these spatial criteria in mind, a search algorithm was implemented in order to search through the lines returned from the PHT algorithm in order to identify the lines corresponding to the sides of the tool shaft and then subsequently the end points closest to the tool tip. The initial implementation followed these steps; for any given video frame:

- The frame is converted into a grayscale image.
- The grayscale image is blurred to remove noise.
- The blurred image is then subjected to the Sobel edge detection operator.
- The detected edge image is thresholded to represent only strong edge candidates.
- The strong edge candidate image is then provided to the PHT algorithm in order to detect lines.
- The lines from the PHT algorithm are then sorted in order to find the longest parallel lines with a set pixel separation (all taken from the known constraints 4.1). The ideal separation was set to 20 pixels after manual inspection of edge detection images.
- Given the 4 endpoints corresponding to the 2 suitable lines on the tool shaft, the endpoints are sorted in order to find the two points nearest to either the center of the image or the previous location.
- The midpoint of the two closest endpoints is taken as the tool tip location.

Figure 4.4: The steps for localizing the tool tip.

The Hough line data sorting component can be further broken down into a series of checks and comparisons. A ‘for’ loop is used to step through the array of hough lines data returned from the PHT algorithm. Each Hough line is defined by two endpoints and is therefore grouped by (x_1, y_1, x_2, y_2) . Hence each time the PHT algorithm is run, an array of endpoint groupings is returned. For each endpoint grouping, a nested ‘for’ loop then queries all the other Hough lines in the array in order to compare two lines at a time. For each pair of these Hough lines, first the line separation is calculated according to equation 4.1. Next, the deviation of line separation from the ideal pixel separation is calculated. For the experimental camera setup, this separation was determined to be 25 pixels. However for different cameras or different ‘zoom’ factors, this separation needs to be manually adjusted.

Following this the Θ values (line orientation) are computed for each line in the pair and the difference in Θ is computed. This difference in Θ values is used to score how parallel the pair of lines are geometrically. Next the distance from each of the four endpoints (describing the pair of lines) to the previous known tool tip location is computed using the distance formula. The minimum of those four distances is taken to quantify the proximity to the prior location. Finally the length of each line in the pair is computed and the longer of the two lines is taken as the shaft length. For the first frame, or if a tool is lost, the last known location is switched to be the center of the image. In this way the line ends in the center of the image are reset as the tip.

$$Sep = \frac{\left| \begin{array}{c} X_{2,1} - X_{1,1} \\ Y_{2,1} - Y_{1,1} \end{array} \right| \times \left| \begin{array}{c} X_{2,1} - X_{1,2} \\ Y_{2,1} - Y_{1,2} \end{array} \right|}{Max(length_1, length_2)} \quad (4.1)$$

Where $length_i$ represents the length of line i and $X_{i,n}$ indicates the n^{th} tool tip on line i .

(Note: here \times represents the vector cross product).

Once these 4 metrics (line separation, theta offset, distance to prior, and length) have been computed, the algorithm compares each pair lines with other pairs in the array to identify the most ideal pair. This comparison involves a series of nested logical checks. The first check is if the line separation is within 5 of the ideal separation (25 pixels). The next check is if the theta offset between the two lines is within 0.1 radians of parallel, in other words close to zero difference in angles for the two lines. The next check determines if the proximity to the prior known location is closer than any previously analyzed pairs in the line array. The final check is if the length is longer than any previous analyzed pairs.

If any given pair of lines from the PHT array successfully meets these logical checks, the two indices corresponding to those endpoint groups in the array gets saved to a temporary variable. The length and proximity to the prior location for that pair of lines is also saved to a temporary variable so that any subsequent pairs left in the array can be compared against those standards. Using this series of checks the algorithm is almost always able to identify the longest, parallel lines which are nearest to a last known pixel location.

In the case of 2 or more lines, this logical comparison series is augmented only slightly to find which tools prior location a pair of lines is closest to. This helps categorize the lines into those which are possible candidates for any of the 2 or more tools known to exist somewhere in the image. After the lines have been sorted into grouping for possible tool candidates, then the subsequent length, theta, and separation comparisons are performed.

Once a suitable (x_p, y_p) pixel location is determined for a tool tip in the first frame in a video sequence, that location is saved, along with the orientation and the length in order to use that information to filter data for the next frame. In the search algorithm for frames in which the previous frame had a accurate detection, the PHT line data is filtered so that lines with endpoints closest to the previous known location is preferred over lines with farther away endpoints. This (x_p, y_p) pixel location is also saved so it can be used to compute world (X_w, Y_w, Z_w) locations once compared with the tool location in the other frame. For the purposes of the test program and in order to visualize the tracking once a suitable (x_p, y_p) location for the tool tip was found, a green circle was super imposed over that location on the original image. Initial results proved very promising in that the tool tip is nearly always located within a few pixels of the actual location on the image. As indicated in Fig. 4.5, the tool can be located in several quadrants of the image plane and is not limited to certain locations in the image.

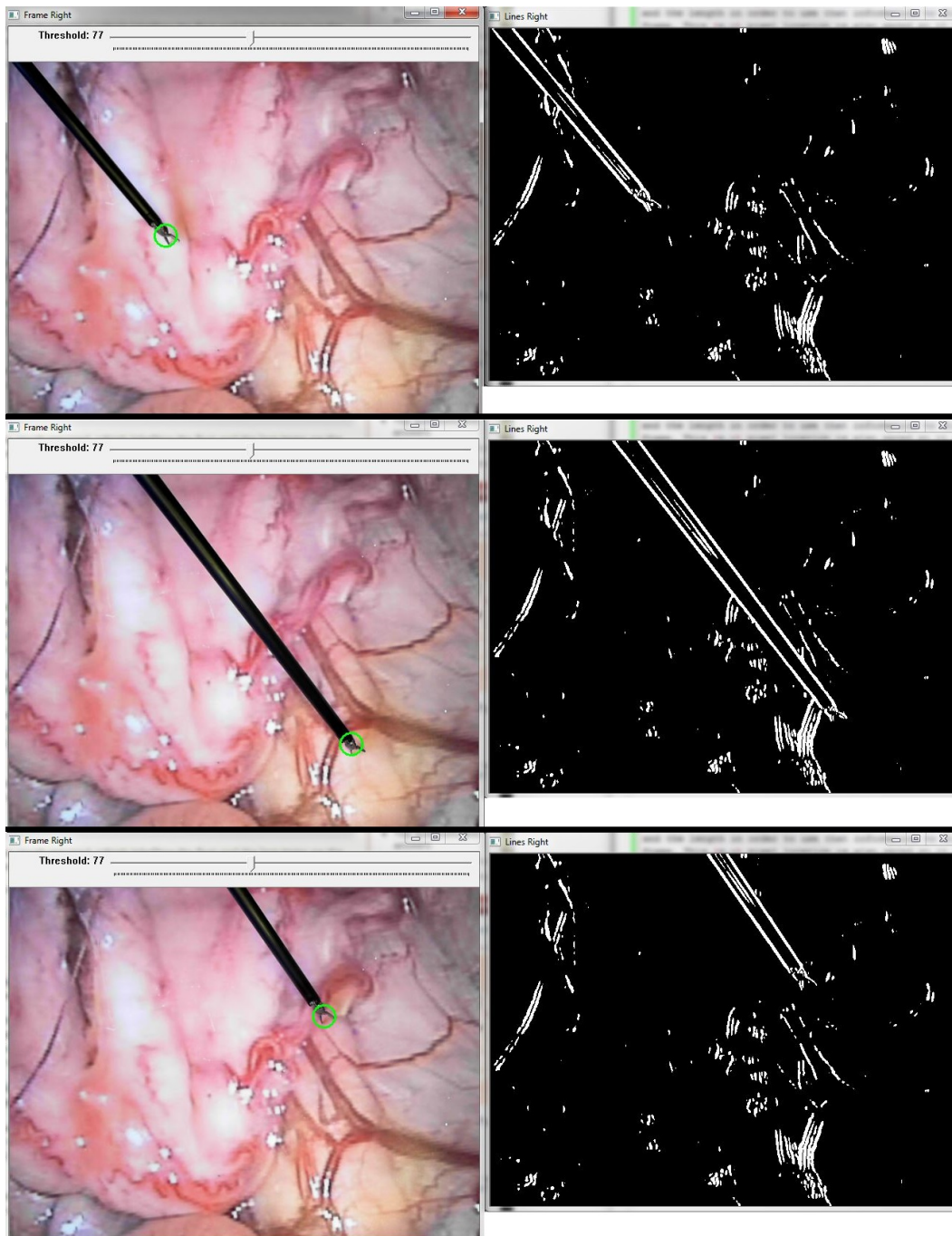


Figure 4.5: A sequence of tool movement with tips circled in green.

Given the demonstrable success of the joint hough and geometric filtering approach to tool detection, a program was written to fully pursue this approach and develop a complete algorithm which can successfully detect and locate 1 or more tool tips in image space and then accurately extrapolate that location into 3D global coordinates using the stereo camera setup. This algorithm requires 5 main high level steps:

- Frame capture (both cameras).
- Edge detection (both channels).
- Hough transform (both channels).
- Data sorting, line selection and endpoint selection (both channels).
- Disparity and global coordinate calculation.

Since the first 4 steps have been explored, implemented and proved effective earlier in this section, the primary step left to implement is the disparity calculation and world (X_w, Y_w, Z_w) coordinate calculation.

4.1.1 Modified Depth Extraction Method and Calibration

In order to extract depth, the object localization in the image will be performed twice (once on each stereo channel). This approach allows the disparity to be calculated only for the individual tool tip locations in each channel. This is used instead of the stereo correspondence method, which computes the disparity for all strong features in the image. This is a marked difference with other approaches outlined in the literature. The full stereo correspondence requires additional computation time that can be avoided by detecting the tool in separate channels simultaneously.

To accomplish this, the program utilizes multi-threaded processing so that each channel can be analyzed simultaneously. The multi-threaded approach is explained in more detail in section 4.2. Once the tool has been localized in both left and right images, the (x_p, y_p) pixel location (with origin at the image center) is saved and the used to compute the disparity and subsequently Z_w depth information. Using this depth information, the (X_w, Y_w) information is computed in the Cartesian registration step (Section 4.1.2)

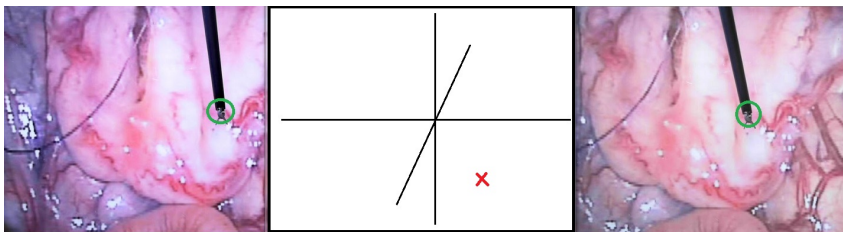


Figure 4.6: The tool is detected in the right and left channels and the 3D location (x) is computed for only the tool tip.

As indicated in figure 4.6, only the 3D position of the tool tip is computed using pixel location and disparity. The rest of the scene is ignored in terms of 3D reprojection. In order to use the pixel location and disparity, a custom calibration method was developed. Instead of following the standard pinhole model calibration [65], [68], which provides a linear transformation, we elected to gather calibration data and develop a model based on the observed relationships. In order to observe these relationships, raw data was collected based on features visible in the right and left cameras.

To use the individual coordinate disparity mapping as opposed to the stereo correspondence method, the stereo camera setup required a modified calibration in order to find a suitable mapping equation from $(x, y, \text{disparity})$ values into (X_w, Y_w, Z_w) global coordinates. In order to accomplish this a custom calibration board was designed with cross hatches arranged in a grid formation with 24.5 mm separation (Fig. 4.7).

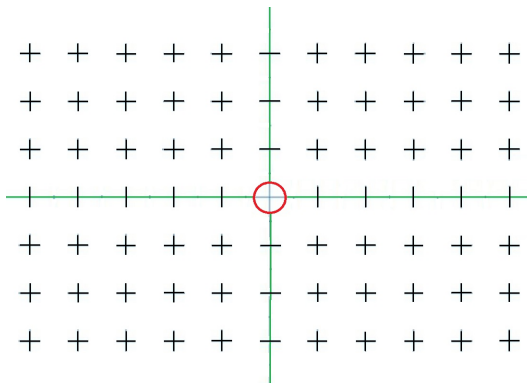


Figure 4.7: A calibration board consisting of a grid of crosshatched marks.

A custom calibration program was written using OpenCV so that the mouse location in a given image could be recorded using a mouse callback function. This callback function allows the user to click on a crosshatch and save that pixel position. The calibration program also uses a series of superimposed rectangle and circles in order to correctly orient the board before calibration begins. The board is placed in the visible field of each camera so that the board is perpendicular to the optical axis of each camera. Then the distance from the board to the camera mount is measured (using a caliper) to get a baseline Z value. This baseline Z value is entered into the calibration program. The center hatch of calibration board is then selected using a mouse click. using the disparity from the two center hatches, the ideal image center is then superimposed as a red circle on each channel so that center cross hatch is centered as much as possible in each channel. Once the board is centered, the four outermost cross hatch marks are selected using the mouse click (Fig. 4.8).

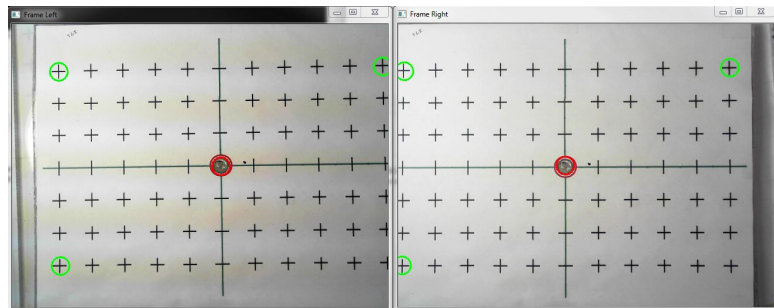


Figure 4.8: Initial Orientation of the Calibration Board.

Once the four outermost hatches are selected, an ideal outer rectangle is superimposed on the screen along with a circle around the center hatch so that board can be rotated and shifted until the four outside corner align with the rectangle corners and the center hatch lies in the center circle for each channel.

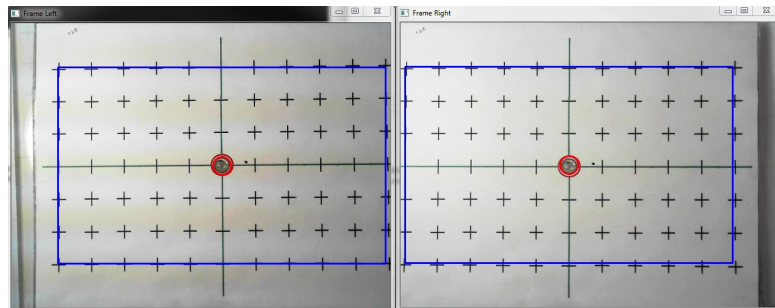


Figure 4.9: Rotational Orientation of the Calibration Board.

Once the calibration board has been centered in each image for a given Z depth, each crosshatch is manually selected in each channel in order from the top left corner across the image. For a given crosshatch in the grid, the x and y indices of that hatch are entered into the calibration program. Then using the known separation (24.5 mm), the actual world (X_w, Y_w) coordinates can be extrapolated. As an example, the (-2,3) index crosshatch corresponds to a world coordinate of (-49, 73.5) mm (Fig. 4.10).

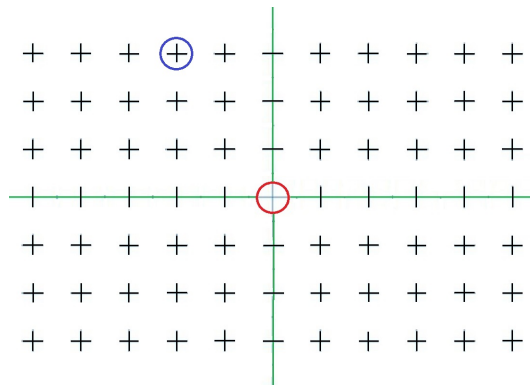


Figure 4.10: In this example the crosshatch index (-2,3) is circled in blue.

Once the crosshatch indices have been entered into the program, the user then selects that crosshatch in both channels using the mouse click function. Each time a pair of hatches is selected from each channel, the separate (x_p, y_p) pixel locations from each channel are recorded along with the world (X_w, Y_w) locations on the board. Both the pixel and world coordinates along with the indices, the computed disparity, and baseline Z are then recorded to a comma separated variables (CSV) file for offline

analysis. Once each crosshatch in the image and the corresponding indices have been selected and recorded. The program can be closed and the camera mount adjusted to a new baseline Z value in order to calibrate for a multitude of depth values.

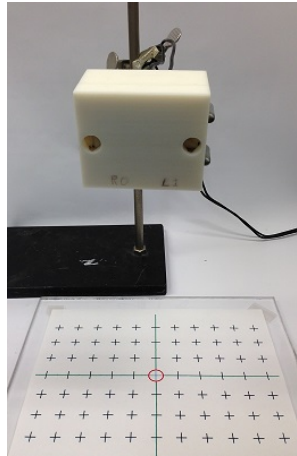


Figure 4.11: The webcam setup was mounted using a chemistry stand in order to vary the depth to the calibration board.

In the case of the calibration routine used for initial attempts with the experimental setup, 5 baseline Z depths were used: 141, 181, 222, 259, and 299 mm. For each of the baseline Z values, the number of crosshatch values recorded depended on how many were visible, for closer depths only 15 or so crosshatches were visible while at the furthest away depths, all 77 crosshatches were visible for data collection. Once all the data was recorded for each crosshatch at each baseline Z configuration, the CSV data was imported into Excel (Microsoft, Redmond, WA) in order to visualize the data and explore different models which could transform disparity data into depth data.

Using the pixel coordinates, world coordinates, disparity and depth data, the relationships between $(x_p, y_p, disparity)$ pixel information were plotted against the corresponding Z_w depth coordinates. The plots between pixel space and global space were used to develop a custom model for transforming disparity into depth. It was immediately obvious that the baseline Z value did not depend only on disparity, but also the (x_p, y_p) pixel position in the image (Fig. 4.12). This is visible in the horizontal outliers for each grouping for a particular baseline Z. This additional dependence is due to the

effects of radial distortion in the camera lens.

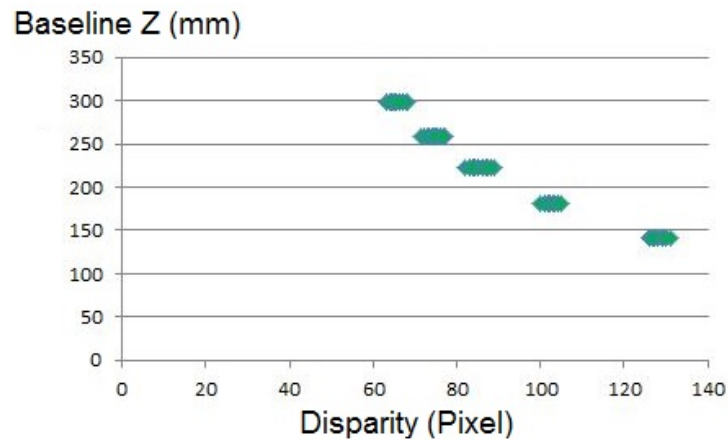


Figure 4.12: Baseline Z is plotted against computed disparity.

Both an exponential decay and a power function were fitted to this data. The best fit equation for the exponential decay had the form $Z_w = 634.34 * e^{-0.012 * disparity}$ with $R^2 = 0.9858$. However the power function equation had an accuracy of $R^2 = 0.9946$ using the form $Z_w = 31992 * disp^{-1.117}$. Upon visual inspection of the graph it is also clear that the power function better represents the relationship (Fig. 4.13). Given the better fit of the power function equation, this model was retained for further model characterization with the inclusion of pixel location dependence.

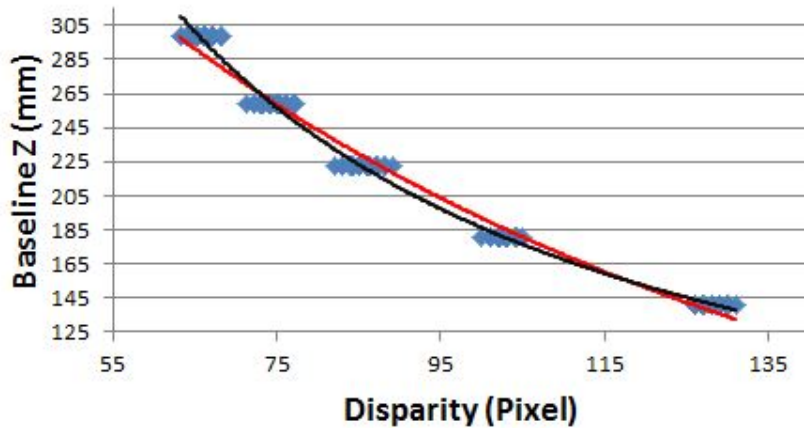


Figure 4.13: The exponential decay line is shown in red and the power function in black.

Using the initial model equations for disparity and Z_w from excel, a Matlab (Mathworks, Natick, MA) script was developed in order to further refine the transformation equations and incorporate x_p and y_p into the model for Z_w . In order to do this, the built in non-linear fit function ‘nlinfit’ from Matlab was used to best fit the model for Z_w given the data from the calibration sequence. Given that the (X_w, Y_w) models both depend on Z_w , priority in this work was given to the Z fitting. The desired model given for Z_w had the form:

$$Z_w = a_1(\text{disparity}^{a_2}) + a_3x_p + a_4y_p + a_5 \quad (4.2)$$

$$A_z = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \quad (4.3)$$

Here a_i represents a coefficient for each term that will be determined using the non-linear fit function. The non-linear function also requires initial guesses for each of the coefficients to be determined. For these initial guesses, the data from the previous linear regressions was used for the first two values and the last 3 coefficients were based on

the error in the linear regression fit. The coefficient guesses used are set as:

$$A_{z,guess} = \begin{bmatrix} 31992 \\ -1.117 \\ 0.0081 \\ 0.0256 \\ 0.3 \end{bmatrix} \quad (4.4)$$

The non-linear regression on the specified model and the calibration data yielded the following fit:

$$A_{z,fit} = \begin{bmatrix} 24939.2089 \\ -1.046456 \\ -0.00799064 \\ -0.02548524 \\ -14.71241555 \end{bmatrix} \quad (4.5)$$

This model was retained for evaluation in conjunction with the linear fit models for X_w and Y_w . It is important to note that the Z_w depth information is necessary in order to subsequently compute the Cartesian registration.

4.1.2 Cartesian Registration Calibration

Once a suitable mapping between $(x_p, y_p, disparity)$ and Z_w had been identified and the coefficients calibrated, the depth value could then be used to register the Cartesian coordinates in global space. The same calibration data from section 4.1.1 was also used to characterize the relationship between pixel location and world (X_w, Y_w) location. Upon inspection it is obvious that there exists no pure relationship between pixel x_p and world X_w because at different depths, disparity is magnified between the channels and each x_p pixel contains a wider area of world space. However, using the Z data calculation it was hypothesized that world X_w could be calculated as a linear relationship to the product of world Z_w and pixel x_p . Upon visual inspection of this plot, it is apparent that this relationship is represented perfectly well as a linear fit (Fig. 4.14) and the form was determined to be $X_w = 0.0016 * (x_p * Z_w) + 0.3643$ ($R^2 = 0.9999$).

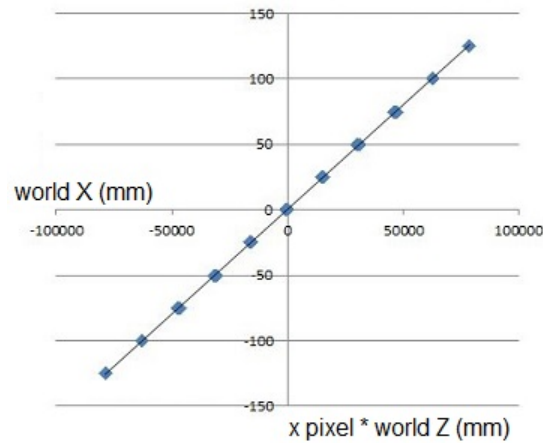


Figure 4.14: A linear fit is applied to the plot of $x_p * Z_w$ vs X_w .

Given that the x_p location was correctly mapped into world X_w coordinates with the additional dependence on Z_w depth, the logical extension was that y_p could be successfully mapped in the same fashion. Again using the product of world Z_w and pixel y_p a plot was fitted with a linear function with coefficients: $Y_w = 0.0016 * (y_p * Z_w) + 0.0014$ ($R^2 = 0.9998$). Again a visual examination of the plot and fit indicates that such a linear model adequately represents the relationship (Fig. 4.15).

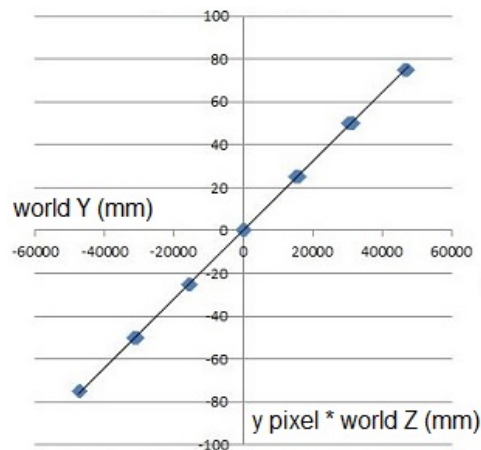


Figure 4.15: A linear fit is applied to the plot of $y_p * Z_w$ vs Y_w .

Using the 3 models and parameters determined by regression, an effective method of

transforming $(x_p, y_p, disparity)$ into (X_w, Y_w, Z_w) was developed. The Cartesian registration that accomplished this was saved as individual functions so that the calibration data could be tested to see if the (X_w, Y_w, Z_w) values mapped from the pixel values accurately represented the true world values. Using the following 3 equations (Eq: 4.6, 4.7, 4.8).

$$Z_w = 24939(disparity^{-1.0464}) + (-0.00799)x_p + (-0.0255)y_p + (-14.71) \quad (4.6)$$

$$X_w = 0.0016(x_p Z_w) + 0.3643 \quad (4.7)$$

$$Y_w = 0.0016(y_p Z_w) + 0.0014 \quad (4.8)$$

The values of (X_w, Y_w, Z_w) were computed using $(x_p, y_p, disparity)$. The Z_w values had to be computed first so that the reprojected values could be used to calculate the (X_w, Y_w) since their errors will compound in actual practice. The reprojections were graphed in Matlab and manually inspected in order to determine the accuracy of the model equations. For the majority of the calibration data, the reprojected points matched up with the real world points to within 1 or 2 mm (Figs. 4.16, 4.17, 4.18). The depth of these seemingly more accurate reprojections are the 3 closest baseline depths used (141, 181 and 22 mm).

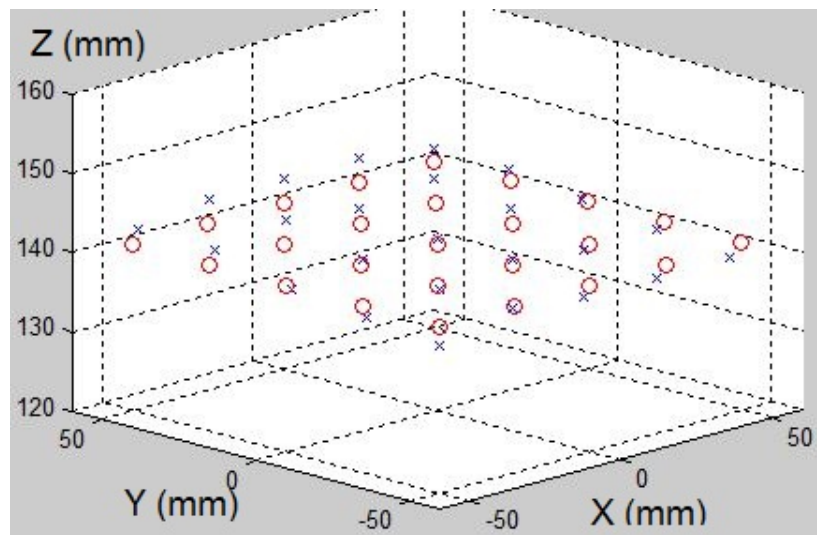


Figure 4.16: Calibration data reprojected at 141 mm.

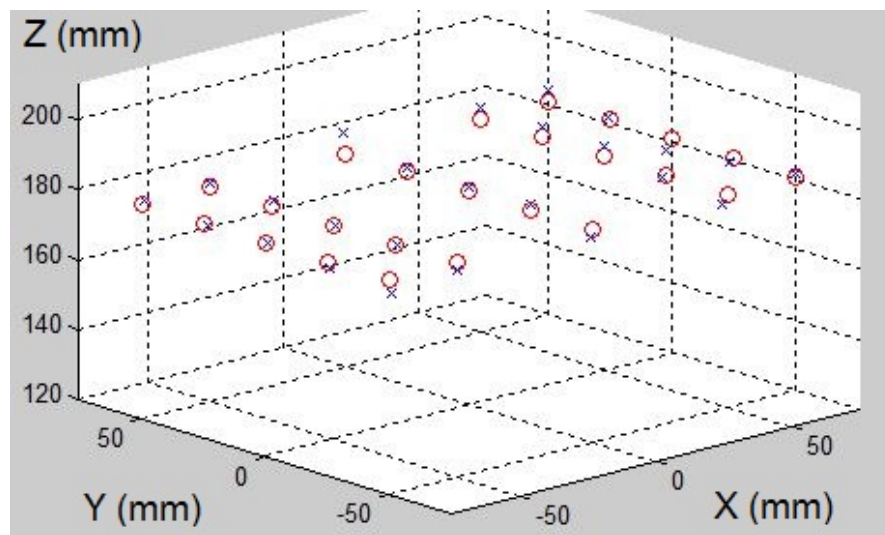


Figure 4.17: Calibration data reprojected at 181 mm.

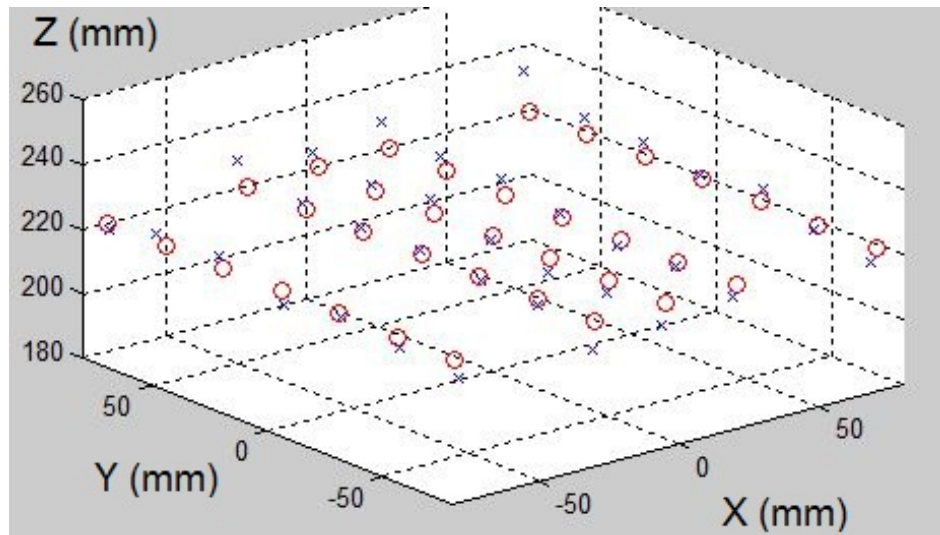


Figure 4.18: Calibration data reprojected at 222 mm.

For the two longest baseline depths (259 and 299 mm) the data appears to be less accurate especially in the Z direction (Figs. 4.19, 4.20, 4.21). It is unclear why the data from longer depths is less accurate, it may be a result of the mouse click accuracy for more distant objects which would imply that another more careful calibration routine would yield better re projections.

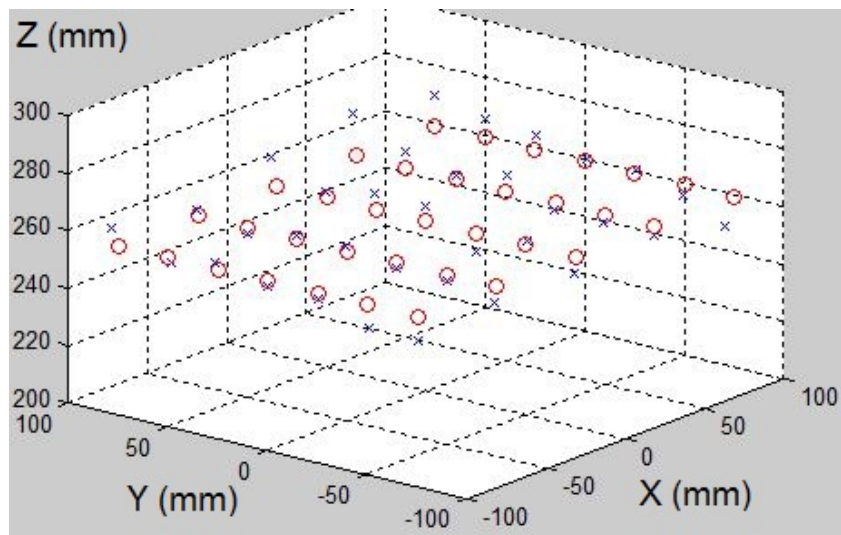


Figure 4.19: Calibration data reprojected at 259 mm.

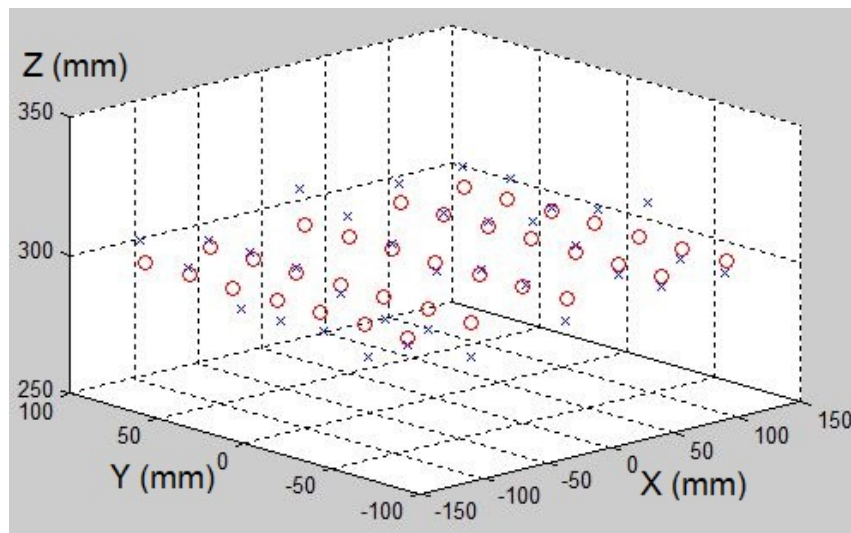


Figure 4.20: Calibration data reprojected at 299 mm.

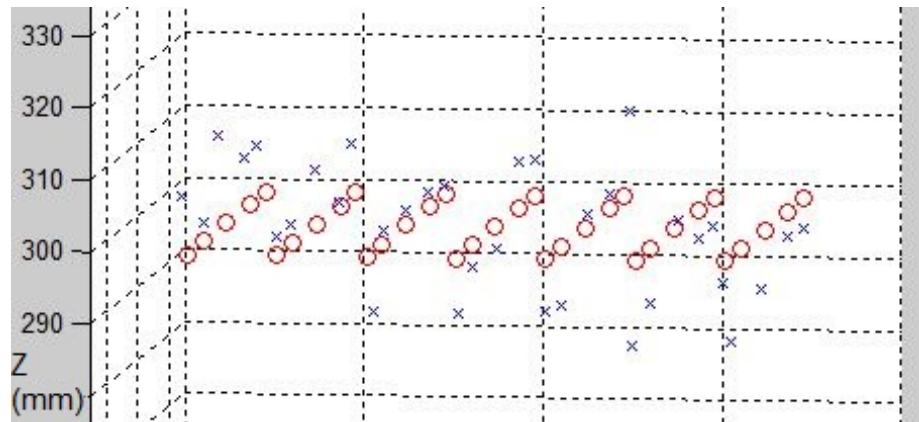


Figure 4.21: Alternate view of the reprojected data at 299 mm.

Apart from the few poor reprojections, the data indicates that the model used is very close to correct for the experimental camera setup. The average reprojections errors for the model is **4.09 mm**. It is worthwhile to note that by observing the calibration data relationships, the disparity-depth mapping is not a linear fit and so the pinhole model would have introduced errors by trying to map a linear model to a non-linear function. It is also beneficial to note that this calibration routine is functional enough to be used with the da Vinci camera feed for OR environment calibration and testing. It is assumed that a similar model for (X_w, Y_w, Z_w) will be appropriate for the da Vinci camera feed as well.

Given the success is using the joint hough transform-geometric approach for detecting the tool in image space and revamped model for projecting pixel coordinates and disparity into real world coordinates, a full optimized program was written to solidify the work. The full algorithm setup and data flow are explained in detail in order to demonstrate both the areas where computational expense has been limited and where bottlenecks could not be avoided.

4.2 Software Design and Implementation

Using the joint Hough-geometric algorithm explained in section 4.1, a fully optimized and multi-threaded program was designed in order to perform more rigorous testing in terms of speed and accuracy. As previously mentioned this software is written in

C++ and utilizes third party libraries for threading and data streaming. This software incorporates the OpenCV, OpenGL and Qt libraries. The application uses the working title ‘SurgicalToolTracking’ and can be run from any computer possessing the necessary libraries and hardware. A high level overview of the software implementation is given here, further details can be found in appendix A.

In the context of this work, the term ‘threading’ or ‘multi-threading’ will refer to the Windows specific allocation of computer processing. This allocation allows a computer to run multiple processes in separate pipelines or ‘threads’. While these threads still need to share the overall central processing unit (CPU) power, the use of threads generally helps prevent bottlenecks in data flow since once one process is done analyzing data it can pass the results to a secondary thread and can then start analyzing a new set of data (Fig. 4.22). These threaded functions run indefinitely until they explicitly exit themselves or are forced to exit. Once all the events from the various threads have been collected, the program can exit safely.

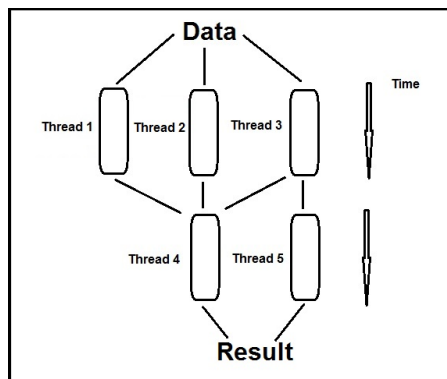


Figure 4.22: In multi-threaded applications, data is analyzed by various threads simultaneously and the results are passed to subsequent threads.

The software framework consists of 5 main components, each component completes one high level task and then transfers data to the next component. The first component is the entry point to the program, the ‘main.cpp’ file. The second component is the video capture function; ‘CaptureClass.cpp’. The third and fourth components are the object detection algorithms for the right and left stereo channels, housed respectively in ‘TrackToolRight.cpp’ and ‘TrackToolLeft.cpp’. The fifth component utilizes the pixel

location data from the object detection threads in order to reproject the tool position in world (X_w, Y_w, Z_w) coordinates. Each of these 5 components is handled in its own thread and utilizes a first-in, first-out (FIFO) information buffer in order to send data to subsequent threads, this buffer will be discussed in detail later. There exist two additional auxiliary functions which offer secondary debugging and analysis functions. These additional components include an options menu graphical user interface (GUI) (for changing variables in real time Fig. 4.24) and a 3D plotting window for visualizing the calculated world position of the tool tip (Fig. 4.23).

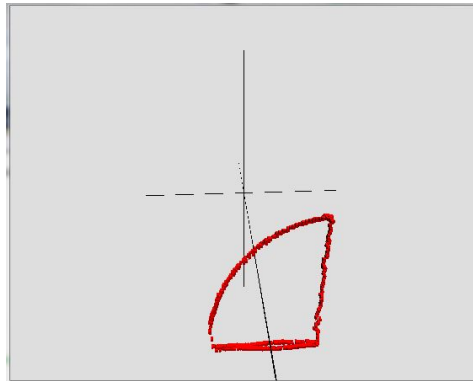


Figure 4.23: World coordinates plotted in real time using OpenGL.



Figure 4.24: Menu GUI using the Qt library.

Another important note is the application uses time stamps important for calculating computational performance and benchmarking frame rates.

4.2.1 Main Entry Point Thread

The ‘main.cpp’ file serves as the entry point to ‘SurgicalToolTracking.exe’ application and is comprised of the ‘Main()’ function. The ‘main()’ functions primary directive is setting up the other application threads and allocating shared memory.

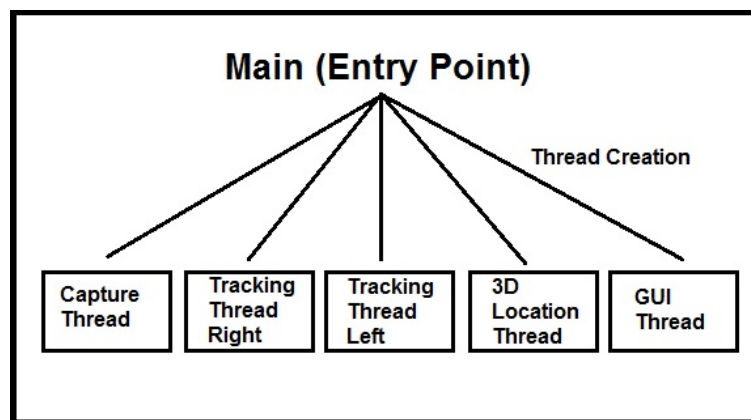


Figure 4.25: The main thread is used to initialize supporting threads.

The ‘main()’ function is also responsible for declaring and initializing a series of buffers for sharing information between threads. These buffers use custom data structures or ‘structs’. These data structures are primarily used to pass image frames and (x_p, y_p) location data (Fig. 4.26). In the case of this work two unique data structs are utilized. The first struct type labeled ‘MultiFrame’ is used to pass the stereo channel frames to the right and left tracking threads. The second struct type is entitled ‘LocateFrame’ is used to pass location and timing information from the tracking threads to the 3D localization thread.

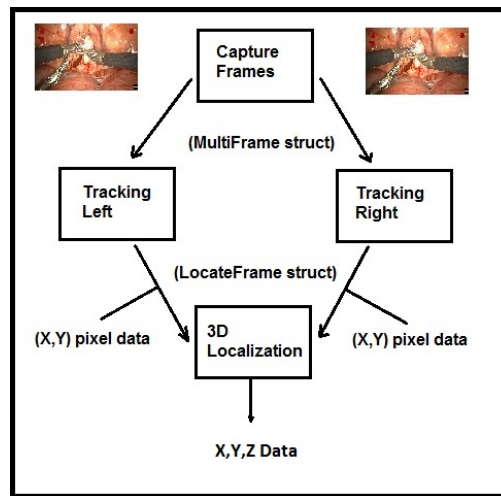


Figure 4.26: The overall flow of information using Data structures.

These buffers are passed to the 4 corresponding threads so that a particular thread can either stuff information into the buffer or extract data out.

4.2.2 Video Capture Thread

The first supporting thread is the video capture thread which opens up 'Capture-Class.cpp'. This threads primary purpose is to open up a video capture device such as the experimental webcams or the da Vinci endoscopic cameras. Once this video capture is open, the capture thread is only responsible for capturing the most recent frame and packing it into a 'MultiFrame' buffer as described in section 4.2.1.

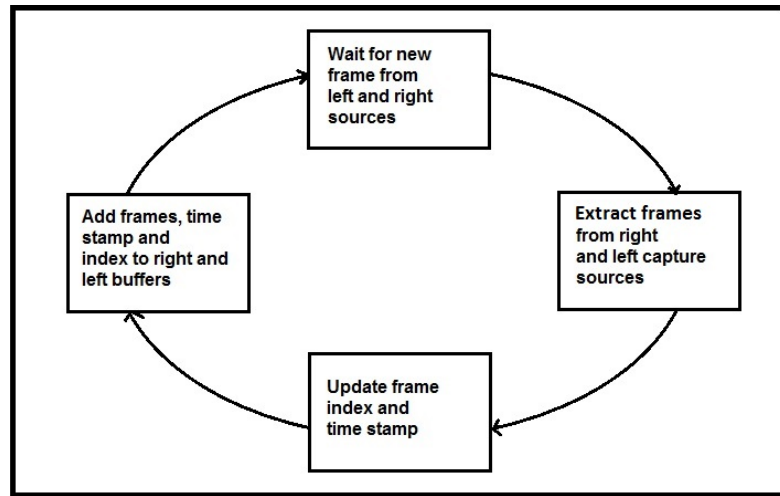


Figure 4.27: The data flow within the capture thread.

The video capture sources are initialized and are specified as the right capture and the left capture. Once the right and left video capture sources are initialized the Frames per Second (FPS) and frame resolution are specified in case the camera sources have been changed outside of the application. The working FPS has been set to 20 so as to not overflow the frame buffer and the resolution has been set to 640x360 resolution but is easily adjustable. While the capture thread is running the right and left video capture objects are constantly queried until a new frame arrives (one every 50 ms at the current frame rate). Once both capture objects retrieve a new frame, a frame index is incremented by one. The same frame index is grouped with both the left and right stereo frames. This index is later used to keep track of frames and ensure left and right frames captured at the same time are analyzed together. Next the time stamp is updated so that later threads also know the time at which that particular frame was captured. The frame and index information is then passed to the right and left tracking threads (Fig. 4.28).

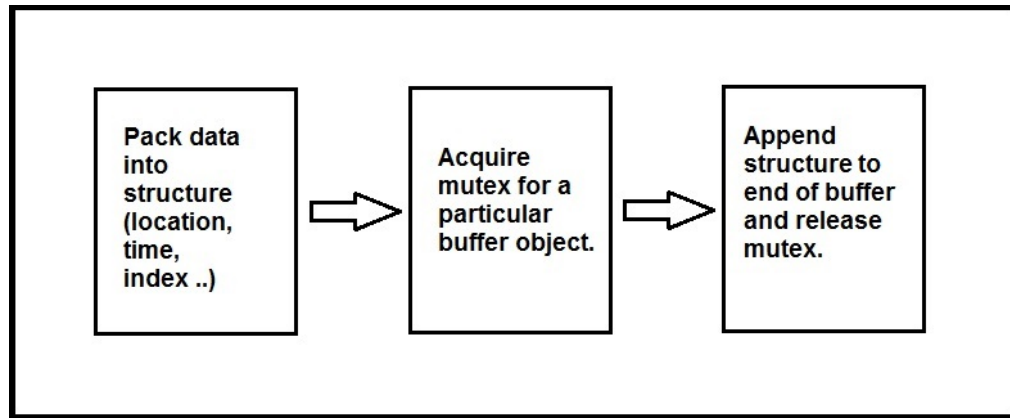


Figure 4.28: Event sequence for adding data to the buffer.

4.2.3 Right and Left Tracking Threads

Once the capture thread described in section 4.2.2 has captured a new pair of stereo images, the right and left frames are then sent to their respective tracking threads. The primary purpose of the two tracking threads is to implement the tool tip object detection algorithm developed in section 4.1. The object detection is separated into two separate threads so that object detection can be performed simultaneously on each stereo channel for any given pair of frames.

While the tracking threads are running, three main tasks are performed; extracting the oldest frame from the 'MultiFrame' buffer, performing the object detection algorithm on that frame, and appending the (x_p, y_p) location data for all tools found into a buffer.

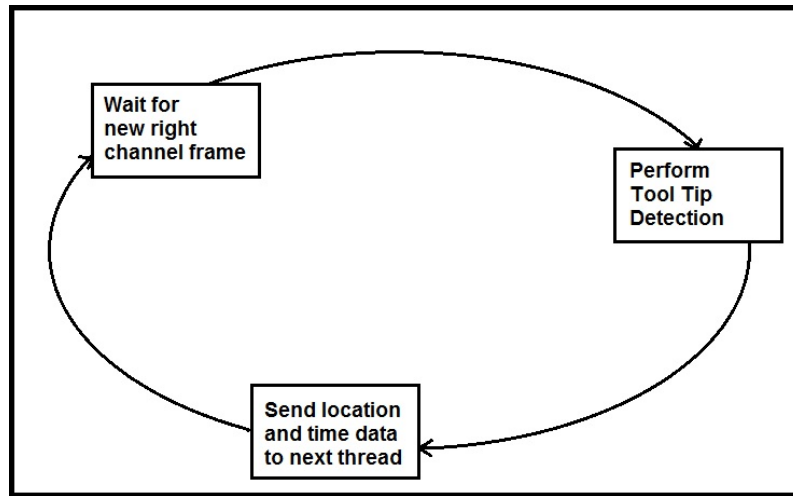


Figure 4.29: The flow of information in both left and right tracking threads.

The oldest object in the buffer is acquired and then deleted from the buffer. The index and time stamp corresponding to this frame are stored so that they can be transferred to the 3D localization function.

Next the Hough object detection algorithm is applied in order to find the likely location of each tool tip. This algorithm follows the same steps found in list 4.1. Once an (x,y) location has been determined for all visible end effectors, the position data is sorted so that each tool tip is labeled according to its proximity to the last known location. This means that tool 1 is labeled as tool 1 in the current frame if its position is the closest to the known location for tool 1 from the previous frame. The position data for each tool tip is filtered using a high-pass filter of size 5 (Eq: 4.9). This filter allows slight noise in the pixel location to be mitigated with previously known locations while still placing the most confidence in the most recent location. Using this filtered pixel location, a circle is then superimposed on the current frame for purposes of visualization. If a particular tool becomes occluded or leaves the field of view, that tool is considered lost and that tool location turns red on the visualization. A lost tool instance is also then passed to the localization thread.

$$X_f = 0.3667 * X_n + 0.2833 * X_{n-1} + 0.2 * X_{n-2} + 0.1167 * X_{n-3} + 0.0333 * X_{n-4} \quad (4.9)$$

Finally, the pixel location, along with time stamps and indices are passed in a buffer to the localization function. The frame index passed is the same index taken from the frame buffer which is set in the capture thread. The buffer information passed also includes a value indicating if and which tools have been lost.

4.2.4 3D Localization Thread

The final step in the tool tracking algorithm involves taking pixel information about the location of each tool tip and transforming this information into 3-Dimensional world coordinates. The 3D localization thread effectively implements both the depth extraction and Cartesian registration steps simultaneously. After the two tracking threads (right and left) described in section 4.2.3, the pixel location (x_p, y_p) of all tools are then passed to the localization thread.

The 3D localization thread is responsible for implementing the transformation equations which map $(x_p, y_p, disparity)$ information into (X_w, Y_w, Z_w) data. This mapping follows the same equations determined in section 4.1.1. Specifically, this thread implements equations: 4.6, 4.7, and 4.8. In order to compute the world coordinates of each tool. Once this information has been calculated, this threads other main responsibility is to output all location and timing data to a data log.

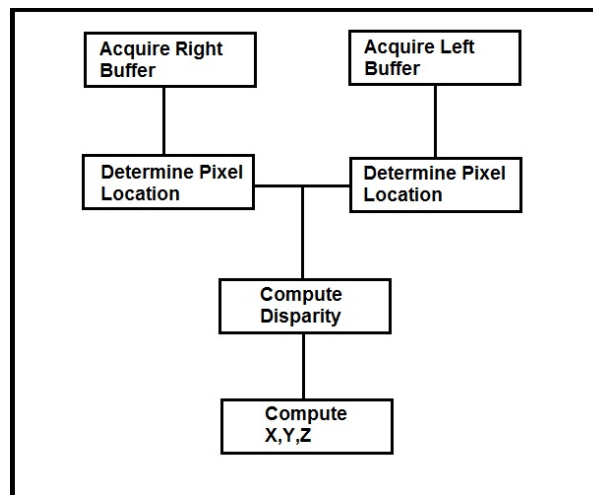


Figure 4.30: The flow of data in the localization thread.

The location information from the buffer is first analyzed in terms of frame index. The frame index from the right and left tracking threads are compared to ensure that the same frame index and in turn the same frame are being analyzed together. Once the thread has acquired new location information from both the right and left buffers and the frame indices have been checked, the (x_p, y_p) tool locations are checked to ensure the tool has not been lost. In the case of a particular tool being lost in one or both channels, that tool's 3D location is simply not computed. Using the pixel locations, the tool tips are then enumerated so that tool tips in the right channel are paired with the same tool tip in the left channel (Fig. 4.31). When the algorithm first starts or if the tools are lost, this enumeration is reset based on left and right quadrants of the image. The tool on the left side of each channel is labeled 'tool' 1 and the tool on the right side is labeled 'tool 2'.

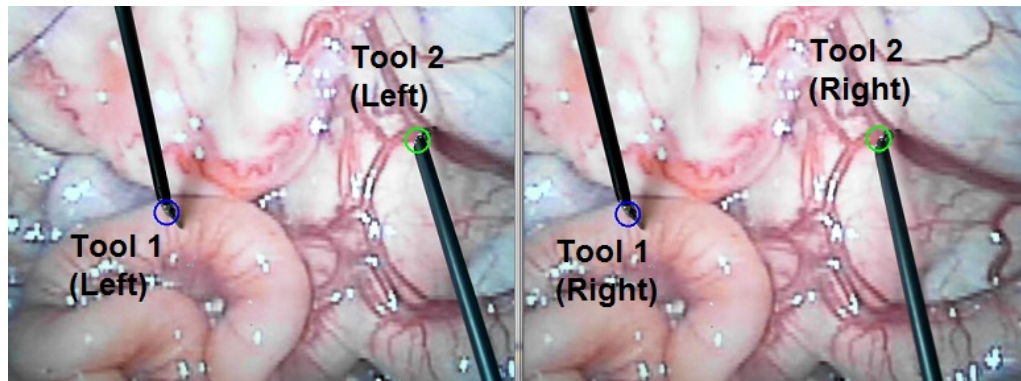


Figure 4.31: Tool tips are enumerated based on previous location.

Next the disparity is computed using the distance equations between right pixel locations (x_{pR}, y_{pR}) and left pixel locations (x_{pL}, y_{pL}) . The disparity is then filtered again using a moving average filter (Eq. 4.10). This filter is required because of the inherent noise experienced when tracking the tool in separate frames. The lines returned from the PHT algorithm may have slightly different characteristics in the right frame than the left frame. This filter helps mitigate that noise.

$$D_f = 0.2 * D_n + 0.2 * D_{n-1} + 0.2 * D_{n-2} + 0.2 * D_{n-3} + 0.2 * D_{n-4} \quad (4.10)$$

After the disparity has been calculated, the equations for (X_w, Y_w, Z_w) can be computed. As mentioned in section 4.1.1, the equation for Z_w must be computed first, followed by X_w and Y_w . These 3 equations must be computed for all tools that are visible in each channel. Finally after all tool locations have been computed the location data along with indices and time stamps are then streamed to the CSV data log.

4.2.5 Conclusion

Using the multi-threaded approach discussed in the previous section, it is possible to attain very high performance tracking using the PHT algorithm and geometric constraints. The hough transform and geometric constraint approach itself is very computationally efficient and only relies on identifying the two edge lines which outline the tool shaft. These two edge lines can then be used to locate the tool tip. The creative use of multiple threads and data buffers allows each sub-task in the tracking algorithm to be performed separately. Single threads need only be used for certain tasks where multiple data sources must come together, such as in the final 3D localization thread (section 4.2.4). The data flow follows the simple movement from video capture to object detection to 3D localization (Fig. 4.32).

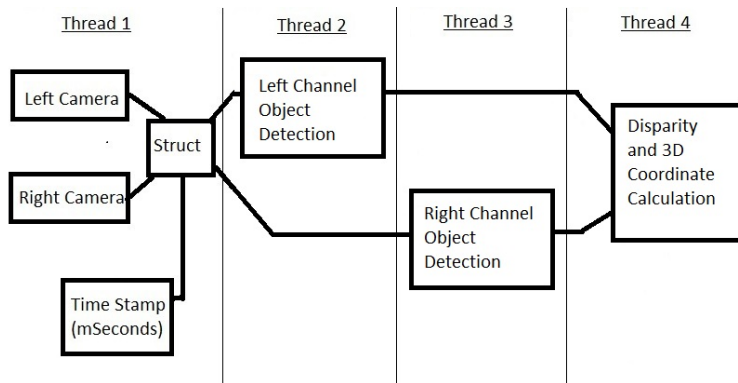


Figure 4.32: An overview of the multi threaded software design.

Much emphasis was placed on optimization in data flow and computational efficiency. Since frame rate was a crucial goal in the development of this tracking program (Table 3.1), not only did the algorithm need to be efficient, but the implementation required creative methods for improved speed. This implementation also ensured other design

goals including tool agnostic tracking and accurate timing data. The tool tracked can be any surgical tool as long as it has a long dark tool shaft. The timing data is provided via the independent timing structures in each thread.

Chapter 5

Experimental Verification and Performance Characterization

5.1 Performance Benchmarking Overview

In order to quantify the performance of the tracking algorithm outlined in Chapter 4, a series of benchmarking routines had to be developed. The primary metrics for which performance of this algorithm will be evaluated are the speed that it takes to track the tool in any given frame and the accuracy therein. This performance metric is gauged in terms of frames per second that can be tracked in the video sequence. Accuracy is calculated by the deviation of the measured tool tip coordinates, in Cartesian space, from known coordinates of a physical calibration board. A third metric is slightly more qualitative; robustness. This metric is evaluated based on what tool configurations result in successfully located tools and which configurations cause lost tools as indicated by manual inspection. Each metric is first evaluated for the experimental camera design, therefore the frame rates at which the camera captures is limited to 30 FPS. In Chapter 6 the metrics and evaluation of the operating room performance are presented.

5.2 Experimental Setup: Computational Speed

This section presents the methods for calculating computational performance for the individual components of the algorithm. Since each component operates in its own

thread, the timing data is calculated individually with each thread and then compiled offline in order to gauge the speed of the total tracking algorithm.

In order to determine the first metric, tracking frame rate, we calculate the time each tracking component described in Section 3.2 takes to complete. For the purposes of this work, the time each component takes to perform its specific tracking function will be calculated as the time difference between when a new frame is received and when the tracking information has been calculated and passed along. This standard does not include the time each component has to wait for a new frame to come in. Time is calculated in units of milliseconds.

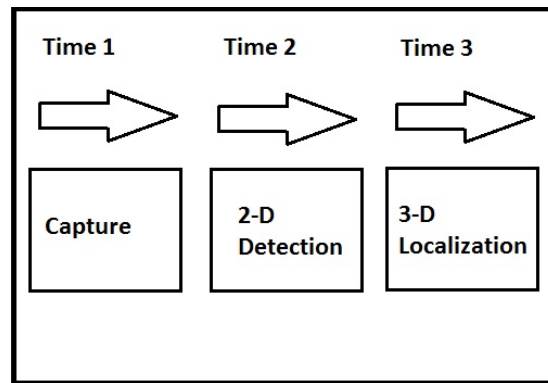


Figure 5.1: The three different components of the tracking algorithm and their individual timing sequences.

For the first component of the tool tracking algorithm, the capture thread, there are no significant time delays. Since this thread simply captures new frames when they come in and passes them along, there are no bottlenecks. The only action that takes any time is capturing the frame from the webcam. This call takes < 0.1 ms and is therefore not considered in the calculation of tracking time. While this thread does not require noticeable computation time itself, it does provide a standardized start time for the rest of the threads to be compared against. Each frame receives a unique and increasing time stamp which is then used to calculate overall time.

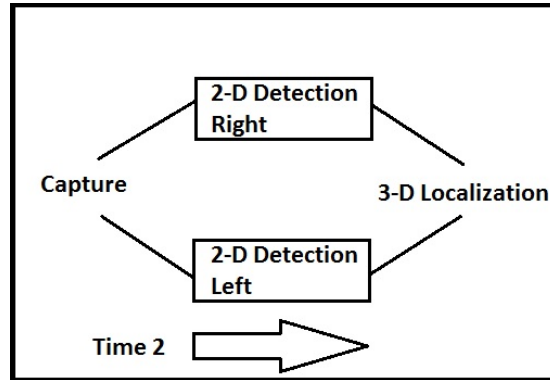


Figure 5.2: The left and right channel tracking threads operate simultaneously.

The second component of the tracking algorithm is the 2-D object detection threads for the right and left channels. As described in Section 4.2.3 the object detection threads are responsible for applying the image functions, the PHT algorithm and then the line sorting function to find the tip of the surgical tool. The time this thread takes to complete is computed as the difference in time between when a new right or left frame is received from the capture thread and when the 2-D pixel location of all tool tips has been found. Since each channels 2-D object detection is performed in a separate thread, the time required by each is not compounded (Fig. 5.2). Since each channel is analyzed simultaneously, we adopt the worst case scenario and consider the time difference to be equal to the larger of the two channels time difference:

$$T_{detect,total}(ms) = Max(T_{detect,right}, T_{detect,left}) \quad (5.1)$$

The third component of the tracking algorithm is the 3-D localization thread. This thread receives 2-D object locations from the right and left tracking threads and uses the location and disparity to perform depth extraction and calculate the tip position in Cartesian space (X_w, Y_w, Z_w) . The timing analysis for this thread is started after the thread receives 2-D location information from each tracking channel. After this the information is used to calculate the global coordinates of the tools in that frame according to equations 4.6, 4.7 and 4.8. After this the coordinate data is saved to the data log and the timing end is signaled. This timing system does not account for the time required for this thread to lock and extract position data from both tracking

threads. This time is also insignificant ($<0.01ms$) and is therefore not considered.

$$T_{tracking}(ms) = Max(T_{detect,right}, T_{detect,left}) + T_{3DLocate} \quad (5.2)$$

Using the longer of the two tracking thread times and summing it with the 3D localization time, we compute the overall tracking algorithm time. Using the overall tracking algorithm time, we compute the frames per second by taking the inverse of the computational time (converted to seconds). This provides an average frames per second that can be analyzed by the tracking algorithm

$$FPS = \frac{1}{T_{tracking}(s)} \quad (5.3)$$

5.2.1 Computational Speed Analysis

Using the methodology explained in Section 5.2, the timing performance was analyzed. The timing data was taken after a working implementation of the working algorithm was initially developed. The timing data was taken with constant lighting conditions and very few additional programs running on the computer system. This ensured consistency in CPU performance. All timing data was performed on the computer system presented in Section 3.7. The experiment was run 10 times, each experiment consisted of the tool tips being moved throughout the visible space while the tracking algorithm computed the Cartesian location. Each experiment lasted several minutes in order to determine an average computation time for each frame. The computational time for each experiment was recorded in order to determine an average computational time and subsequently the frame rate.

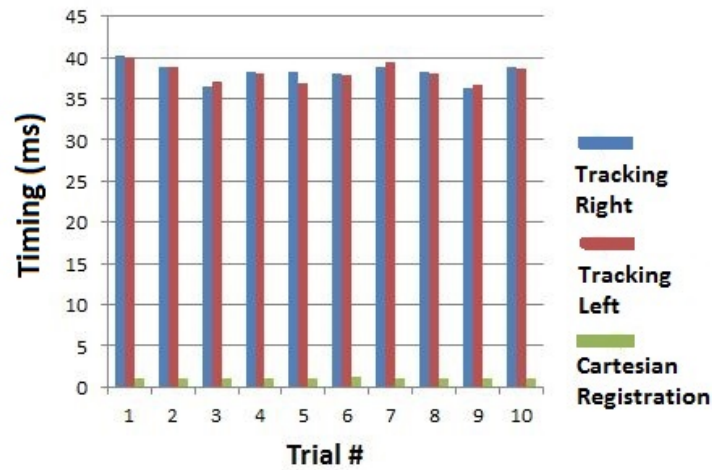


Figure 5.3: The various tracking components and their computational time (ms) over 10 repeated trial data collections.

As seen in Figure 5.3 the computational time of the left and right detection threads is significantly higher than the 3D localization. The object detection threads take around 38 ms to complete. The 3D localization requires approximately 1-2 ms, using the optimized (multi-threaded code). Using the longer of the two tracking thread times and summing it with the 3D localization time, we compute the overall tracking algorithm time.

Using the overall tracking algorithm time, we compute the FPS. This performance was increased using program optimization (Fig. 5.4).

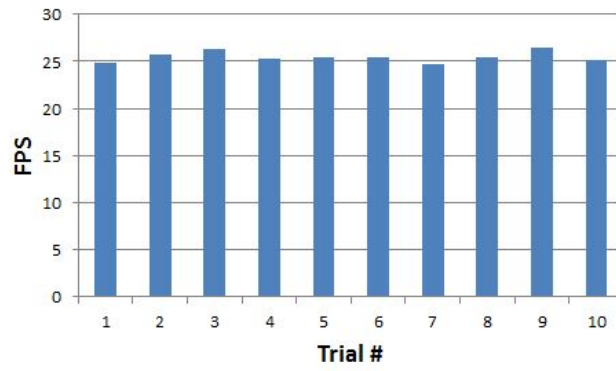


Figure 5.4: Overall Frames per seconds achieved using the optimized tracking algorithm, taken over 10 trial data collections.

Table 5.1: The time in ms is given for the left and right detection threads and the 3D localization thread. Using these thread times, the FPS is computed.

Trial #	Detection Right (ms)	Detection Left (ms)	3D Locate (ms)	Frames Per Second (FPS)
1	40.25	40.0177	1.01	24.839
2	38.898	38.86	0.985	25.708
3	36.494	37.0965	0.9562	26.2793
4	38.3225	38.0423	1.1312	25.346
5	38.3192	36.8055	1.0338	25.411
6	38.117	37.879	1.146	25.4692
7	38.808	39.4107	1.0258	24.7302
8	38.1903	38.066	1.0407	25.49
9	36.2545	36.7886	0.9664	26.4865
1124	38.7749	38.6601	1.0098	25.1353

In total, the computational performance for the entire tracking algorithm proved sufficiently fast. The tracking algorithm presented takes on average a total of **39.9 ms** to complete. The majority of that computational effort comes from the PHT algorithm. This computational time results in an average frame rate of **25.8 FPS**. This computational performance in the experimental setting is satisfactory. It exceeds our design goal of at least 16 HZ for tool motion by 61%. Further more it is a marked improvement over previous attempts, exceeding the fastest reported technique (17 HZ [37]) by 52%.

5.3 Experimental Setup: Tracking Accuracy

The second metric to be characterized is the accuracy with which the tracking algorithm can locate the tool tip in 3D Cartesian coordinates. For this characterization we refer to the coordinate system outlined in Figure 3.11. Using this coordinate system, tracking accuracy is determined by moving the tool tip around a known fixed trajectory via a custom jig. Using the pixel coordinates and disparity found as the tool tip is moved around this trajectory, the world (X_w, Y_w, Z_w) coordinates are calculated. The reprojected world coordinates are then compared with the coordinates of the known trajectory in order to determine the error for each location.



Figure 5.5: The stereo camera is placed above known trajectory.

In order to have a repeatable trajectory with known coordinates, a board was developed using CAD software from PTC Creo (PTC, Lansing, MI). This trajectory board contains 3 circumscribed paths located on the same horizontal plane. Each path takes the shape of an arc connected by two straight lines (Fig. 6.22). Each path is recessed into the board so that the tool tip can rest in the groove and repeatably follow the track. Since the board was designed in CAD and then 3D printed, the geometry of the board is accurately known to within the tolerance of the 3D printer. For this board, a Stratasys Dimension 1200es 3D printer was used (Stratasys, Minneapolis, MN). This printer is capable of resolutions up to 0.254 mm.

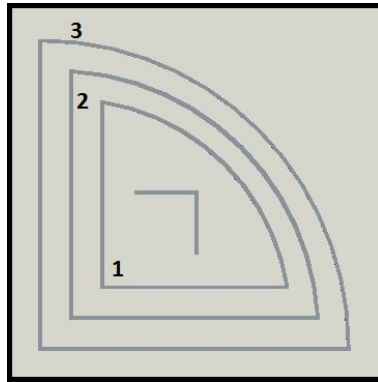


Figure 5.6: The experimental board for tracing a known trajectory, the arc paths are numbered.

In order to use this board effectively, the board has to first be oriented so that the central ‘L’ mark on the board is aligned with the (X_w, Y_w) axes. To accomplish this we utilize an orientation and centering program similar to the one described in section 4.1.1. This program allows the center mark to be adjusted so it is collinear with the center of the two camera stereo setup (Fig. 5.7). Then the corners of the trajectories are rotated so the straight lines are perpendicular to the (X_w, Y_w) axes. This orienting process ensures that the data from the tracking algorithm can be accurately correlated with the known trajectory model.

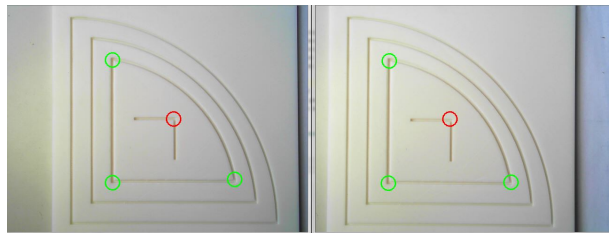


Figure 5.7: The tracing board is oriented relative to the camera.

Using this board, the wrist of the surgical tool is held constant and the tool tip is placed in the recessed groove. Then the tracking application is run in order to locate the tool. The location of the tool for each frame is saved to a data log. While the tool position is being tracked, the tool tip is manually articulated through one of the tracing paths until a full lap around the path has been completed (Fig. 5.8). After the

application has been shut down, the location data from the data log is exported into Matlab for analysis. For a given distance from the camera to the trajectory board, the distance is measured with a caliper and that distance is used as baseline depth Z_w .

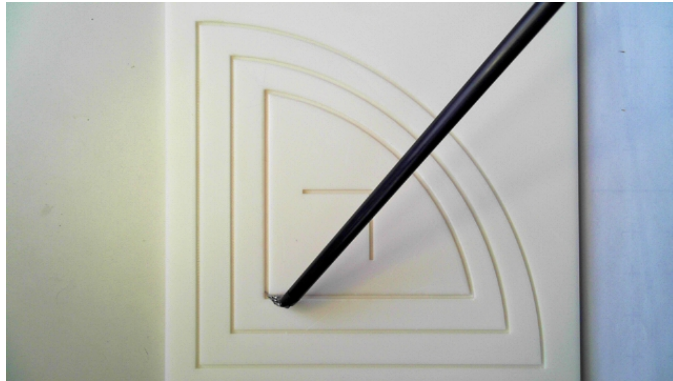


Figure 5.8: The tool tip is traced along the known trajectory.

Using the known geometry of the trajectory board paths, we compile an array of model points (x_m, y_m, z_m) . The model points correspond to locations on the path trajectory. In order to minimize correspondence errors, the separation between model points is set to 0.05 mm. The model points are initially calculated only in the (x,y) plane and the z value for the model is determined by measurement of the distance from the camera to the board. The accuracy testing routine is performed at several different heights.

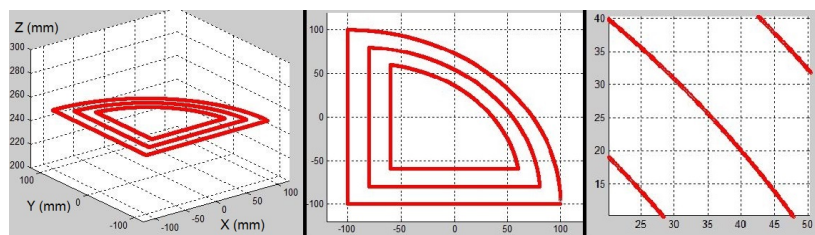


Figure 5.9: Known model locations along the trajectories.

Using reprojected data coordinates and the known model coordinates, we calculate the error through a simple closest point search. For any given data point the closest model point is determined using the Pythagorean distance equation. The distance to the minimum point is then stored and the next data point is analyzed. Using this

system we acquire the error between each data point and the closest corresponding known trajectory point. There is potential for a slight inaccuracy using this method since the model points have finite spacing between one another (0.05 mm). In this case there is a possibility that a data point may actually lie on the trajectory but appear to have a slight error (Fig. 5.10). However, this issue will only result in a small increase of the average reprojection error, and guarantee that the actual error is always less than or equal to the recorded value.

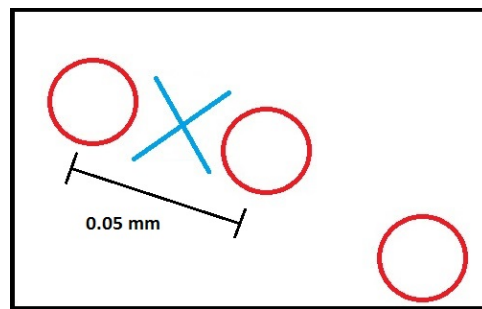


Figure 5.10: The data point (x) lies on the trajectory, but between two model points (o).

The overall noise of the tool reprojection is also quantified. The noise is examined by calculating the deviation of a stationary tool tip over time. In order to calculate this metric accurately, the surgical tool is fixed in a clamp with the tool tip in the visible field (Fig. 5.11). The tool tracking algorithm is then run for an extended period of time and logs tool position. After the tool tracking is finished, the data is exported for analysis. Using the position data, an average location ($X_{avg}, Y_{avg}, Z_{avg}$) is first calculated.



Figure 5.11: The surgical tool is held stationary while tool position is calculated.

Using the calculated average position, the deviation of each reprojected data point from the average is computed. After all the errors have been calculated, an average deviation is determined. This average deviation is considered the noise of the tool tracking algorithm. For any given stationary noise of the tool tracking algorithm, it is not theoretically possible to attain a more accurate overall tracking accuracy. The noise from this algorithm is primarily due to the two stereo object detection channels.

5.3.1 Tracking Accuracy Analysis

Using the experimental design explained in Section 5.3, tool location reprojection data was collected. First the stationary resolution is presented and then the overall tracking accuracy. The resolution provides a good baseline for how accurate the tool tracking algorithm can be for a moving tool.

For both the stationary and moving accuracy, data was collected for a variety of camera distances. The range of distances varied between 200 and 400 mm. The lower bound of 200 mm is required because the whole of the smallest arc of the trajectory board is not visible at closer distances than that. The upper bound of 400 mm was chosen because few robotic procedures have working volumes that large.

Resolution data was collected at a variety of different camera distances and in arbitrary locations in the field of view. The tool tip position was varied so that resolution data could be analyzed for the whole of pixel space. A few of the reprojected data points for the stationary tool are presented from various viewpoints.

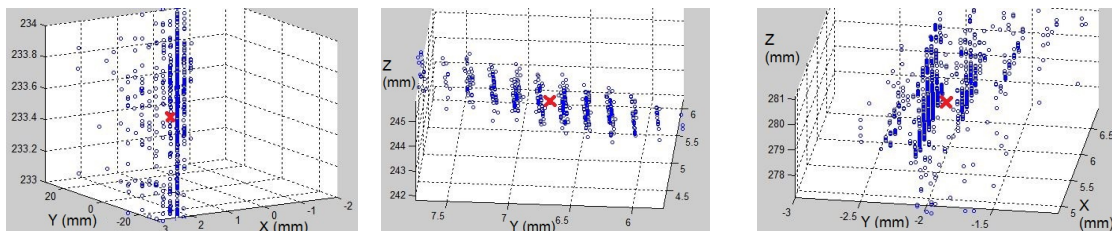


Figure 5.12: The reprojected tool tip positions for a stationary tool at various heights (233, 243, 280 mm).

Figure 5.12 demonstrates the variation in reprojection data from the average location is not excessively large. The majority of the deviations come from the depth variations. This is to be expected from detecting the tools in each channel separately. When the

tool location varies slightly in one channel, the disparity shifts and subsequently the depth changes.

Table 5.2: The reprojected coordinates for the stationary tool are compared with the average positions in order to determine the noise.

Trial #	Depth Z (mm)	X Deviation (mm)	Y Deviation (mm)	Z Deviation (mm)	Overall Deviation (mm)
1	233.6455	0.2316	0.2789	0.5313	0.7223
2	243.814	0.2045	0.4374	0.3236	0.647
3	273.7987	0.3287	0.5184	1.2838	1.5421
4	279.1256	0.1571	0.4295	1.1744	1.4068
5	334.0874	0.8566	0.6133	2.462	2.8338
6	255.9788	0.1817	0.7388	0.954	1.3467
7	251.5063	0.2207	0.2356	0.3609	0.5534
8	353.7072	1.1856	1.299	3.3814	4.1594

The computed noise from the series of trial tracking routines is given in Table 5.2. The average computed noise was **1.6514 mm**. However, if the outlier of the noise data taken at 353.707 mm (4.1594 mm) is removed the average noise is significantly lowered to **1.2932 mm**. It is reasonable to exclude this outlier since the depth of 353.707 mm is near the upper bound of work volumes for surgery. This is a very promising noise characterizations considering the average reprojection error for the depth map calibration was 4.09 mm.

Having computed the average noise, we compute the average reprojection error for the known trajectory tracing. For each height, the specific trajectory used was varied. Again different arcs were used at different heights so as to encompass the whole spectrum of pixel space when reprojecting. Several data sets and the corresponding models are given below in order to demonstrate the performance of the tracking algorithm. The measured data points are given in blue (x) and the model points in red (o).

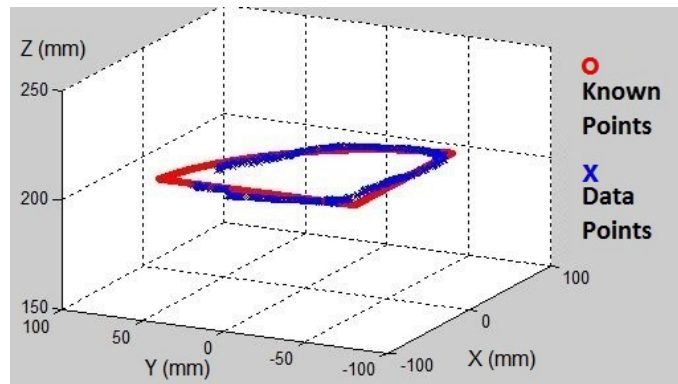


Figure 5.13: Tool motion reprojection for arc path 1 (205 mm).

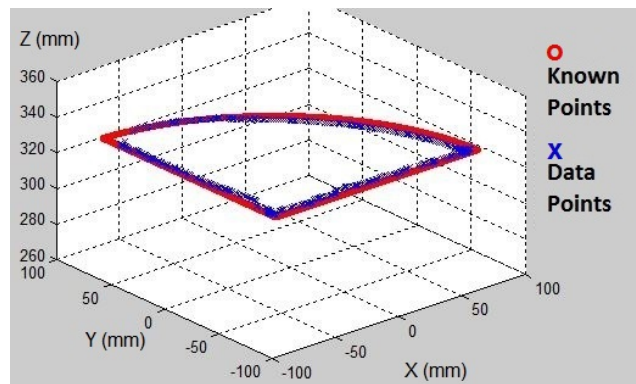


Figure 5.14: Tool motion reprojection for arc path 2 (329 mm).

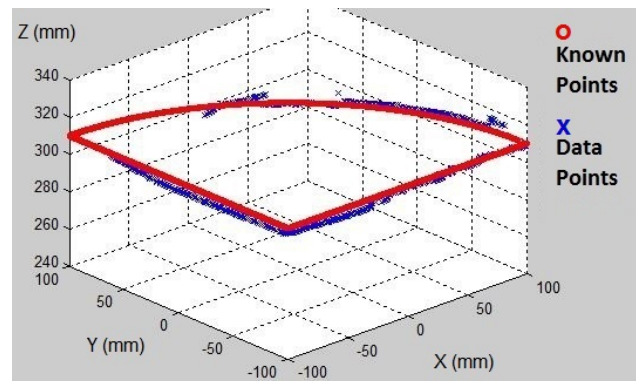


Figure 5.15: Tool motion reprojection for arc path 3 (310 mm).

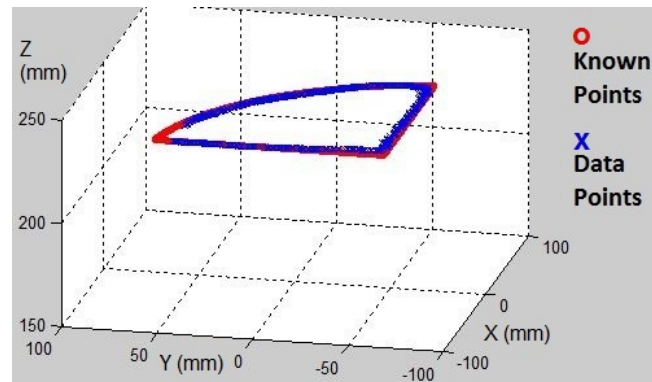


Figure 5.16: Tool motion reprojection for arc path 1 (232 mm).

As indicated by Figures 5.13, 5.14, 5.15 and 5.16, the overall reprojection is qualitatively close to the known trajectory. The (X_w, Y_w) coordinates are correct a majority of the time while the Z_w reprojection seems to fluctuate slightly as the tool is moved around. This is primarily due to small jumps in pixel location of the tool in the two separate channels. A single pixel difference in disparity causes the depth to change by as much as 4 mm (Eq. 4.6). The moving average filter helps to mitigate these fluctuations, however they are not completely avoidable.

Table 5.3 contains a subset the reprojection errors for various baseline depths and sample sizes. The table indicates the baseline depth as well as the average depth in the reprojected coordinates. The data was also analyzed to determine the percent of reprojections where the tool location was within an 8 mm range of the known trajectory. The other ranges analyzed were 2, 4 and 6 mm. Each trial data collection consisted of maneuvering the tool tip through the known trajectory while calculating Cartesian location data. This data was then compared with the known location to determine the error.

Table 5.3: The reprojected coordinates are compared with the known trajectory coordinates in order to determine errors.

Trial #	Baseline Z (mm)	Average Z (mm)	Arc #	Avg Error (mm)	95 th Percentile Error (mm)
1	266	265.86	1	2.53	3.907
2	212	212.877	1	3.33	6.3606
3	289	289.77	2	2.72	5.2641
4	305	305.851	2	2.39	5.1341
5	309	308.636	3	3.968	6.806
6	329	329.43	2	3.396	6.4496
7	280	280.02	2	3.284	5.8367
8	205	205.3	1	3.78	6.6341
9	232	231.9	1	2.18	3.9279
10	274	273.4	2	2.92	4.419

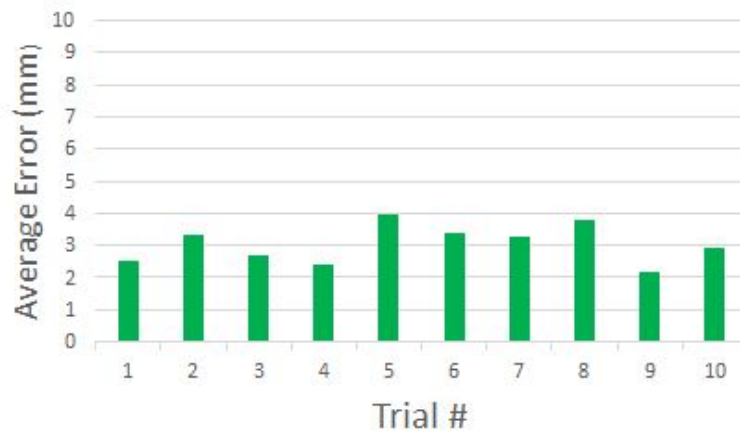


Figure 5.17: The average error (mm) in tool reprojection compared with a known trajectory for several trial data collections.

The errors are reported for the various heights that the tracking was performed at. As indicated, the measured baseline depth (Z) is generally very close to the average reprojected Z value. After compiling the reprojection errors for all trial runs, the average error was found to be **3.04 mm**. The minimum error found was **0.061 mm**. The maximum error was **16.799 mm**. The errors appear to be due to slight fluctuations

in the disparity-depth calculation. In contrast, the average error in the x-y plane was found to be **1.59 mm**. Finally, 95% of the reprojections were within **5.474 mm** of the known trajectory. The next metric that is calculated is the percentage of the time the tool is accurately located within certain ranges of the known trajectory. This metric indicates the ability of the tracking algorithm to correctly identify the tool with a known frequency. The percentage of reprojections within 4 mm is given in figure 5.18.

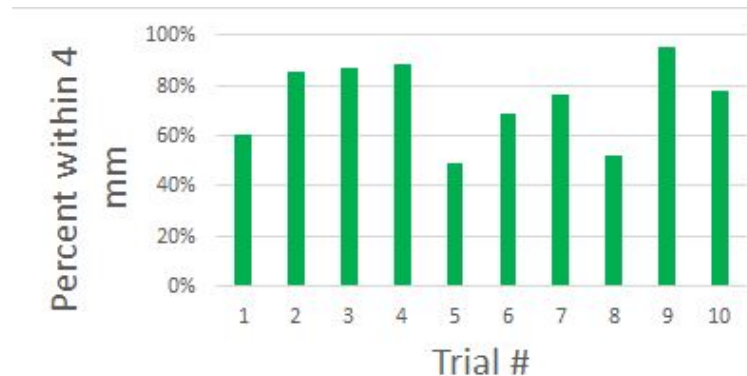


Figure 5.18: The percentage of reprojections within 4 mm of the known trajectory.

The reprojected coordinates for the experimental setup are accurate to within 2 mm of the known trajectory 27.9 % of the time. The reprojections are within 4 mm of the known trajectory 78.8% of the time. The reprojections are within 6 mm 95.26% and within **8 mm** of the known trajectory **99.6 %** of the time. This localization percentage is sufficiently high considering the highest reported localization accuracy of 93% in two dimensions from Reiter et al. [47]. This group reported the 93% tracking accuracy in 2D by manually inspecting the detected tool shaft center axis. Accurate reprojections were counted if the center tool shaft axis aligned somewhere on the physical tool shaft (an 8mm range).

5.4 Tracking Robustness Analysis

In order to determine the robustness of the tracking algorithm, multiple different tool configurations needed to be tested. Additionally, several different lighting conditions and background textures needed to be analyzed. In order to examine these different

scenarios the experimental camera setup was utilized an images were manually inspected for accuracy in order to determine if successful detections were made. Images and analysis are provided in order to depict scenarios in which tool tips were and were not correctly located.

For this analysis two tool tips were tracked. The tool shafts were rotated around the field of view in an attempt to find those configurations where one or both of the tools is lost or incorrectly located. For the majority of the configurations the tool tips are correctly located. A correct localization is determined if the detected tool tip locations in both channels are within 10 pixels of the wrist location. Provided below are a sequence of different tool tip configurations with the tool tip indicated by superimposed, color coded circles in both channels. The right and left channels for each configuration are both given.

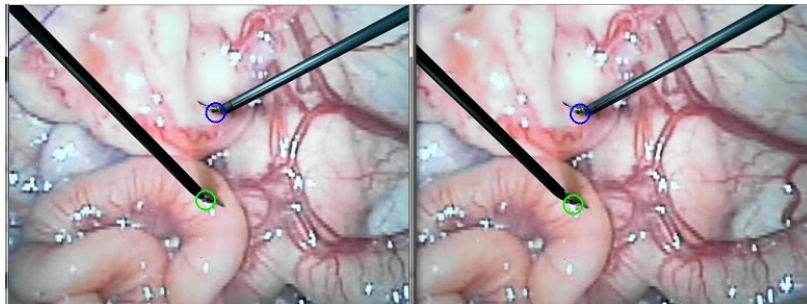


Figure 5.19: Arbitrary tool configuration 1.

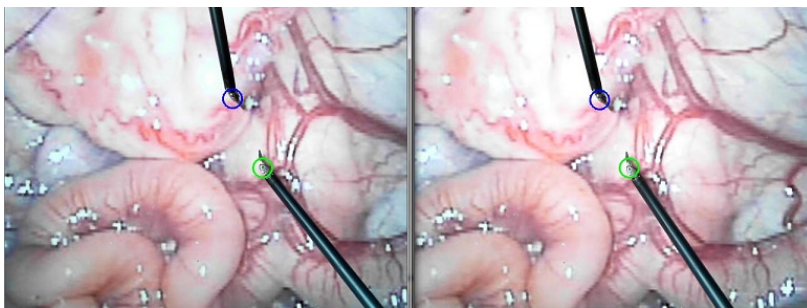


Figure 5.20: Arbitrary tool configuration 2.

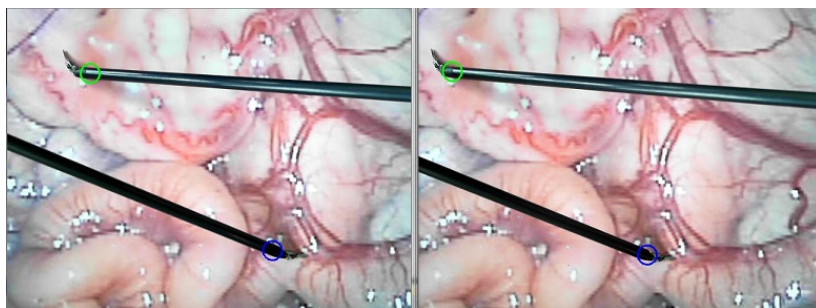


Figure 5.21: Arbitrary tool configuration 3.

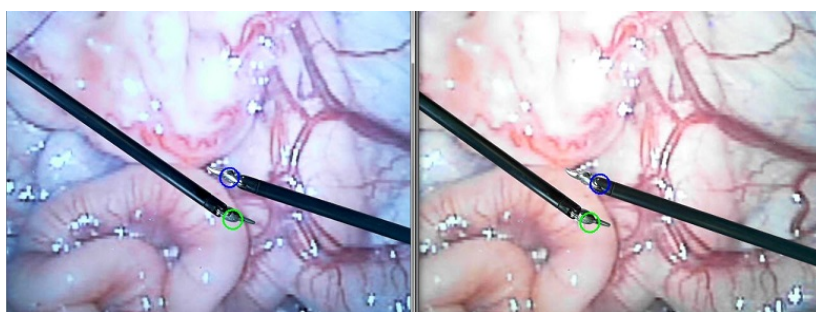


Figure 5.22: Arbitrary tool configuration 4 (Tips in close proximity).

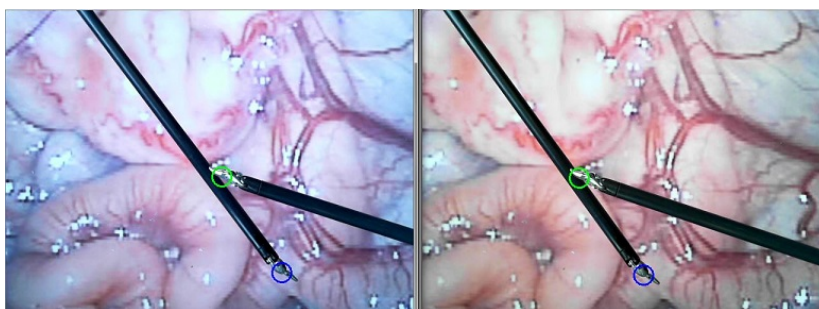


Figure 5.23: Arbitrary tool configuration 5 (Overlaid shafts).

The tool configurations depicted in Figures 5.19, 5.20, 5.21, 5.22, and 5.23 all result in the tools being accurately detected in both channels. Configuration 5 (Fig. 5.23) is especially promising since one tool shaft overlays the other and both tips are still located. There are certain configurations found where the tool tip is not accurately

located, these configurations are depicted in the following figures.

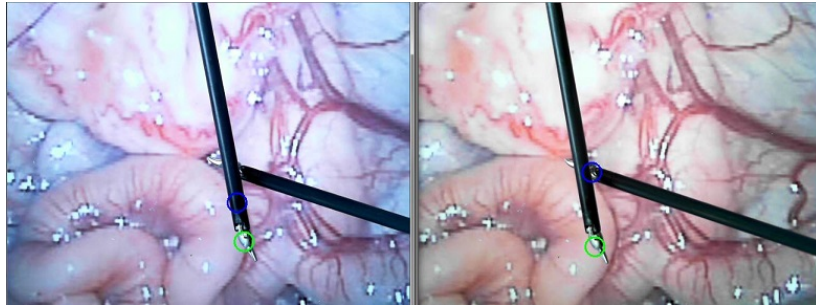


Figure 5.24: Arbitrary tool configuration 6 (Incorrect detection in left channel).

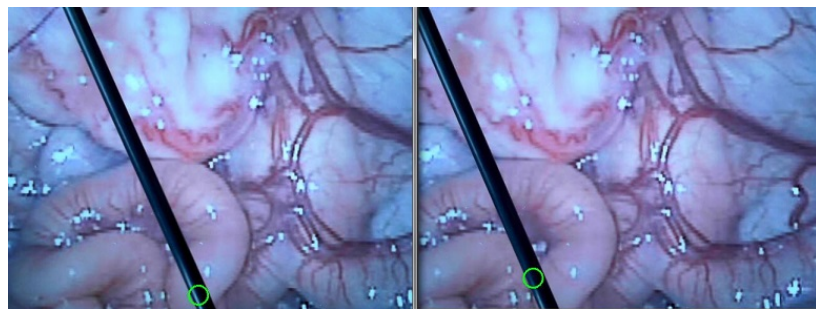


Figure 5.25: Arbitrary tool configuration 7 (Incorrect detection, tool tip not visible).

In the configuration depicted in Figure 5.24, the left tool shaft is occluding the right tool tip. In this scenario the tool tip of the right tool has been incorrectly ‘detected’ along the shaft of the left tool. In the configuration depicted in Figure 5.25, the tool tip is incorrectly located along the shaft and not on the tool tip. This is due to the tool tip not actually being in the field of view. Situations such as this are unavoidable in computer vision approaches as the tool cannot be localized unless it is visible.

A second test was performed to determine the accuracy of the localizations in the event of textures and background objects. To examine this condition, arbitrary objects were placed in the background of the field of view and the tool tracking algorithm was run. While these arbitrary objects would never appear in the surgical site, they do provide an example of varied background textures and objects. The configurations are again depicted along with the tool location superimposed. Most notably, they provide

rectangular geometries intended to confound the PHT approach as an extreme condition of robustness.

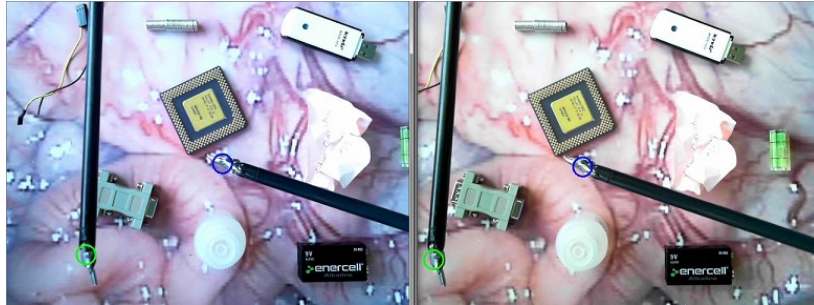


Figure 5.26: Arbitrary tool configuration 8 (Varied background objects).

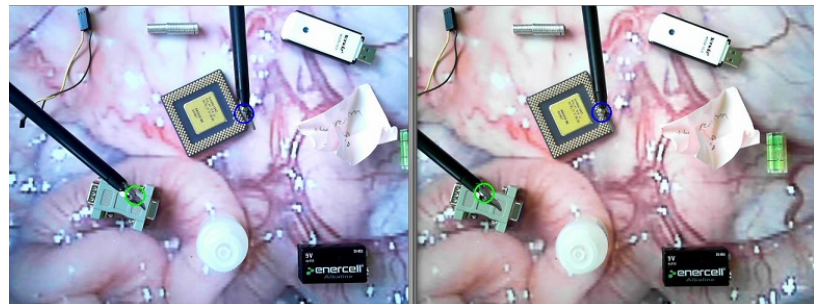


Figure 5.27: Arbitrary tool configuration 9 (Varied background objects).

As indicated in Figures 5.26 and 5.27, the presence of background objects and different textures does not prevent the tracking algorithm from locating the tool tip. The success of the tool tracking algorithm in this setting is attributable to the blurring of the image and the selective geometric features of the tool. The blurring step helps the background objects stay less pronounced in the edge detection. The final analysis for robustness was performed using lighting variations. While the initial tool configurations were performed with a sufficient amount of background lighting, the tool tracking was also examined in dark conditions. These configurations and conditions are depicted below.

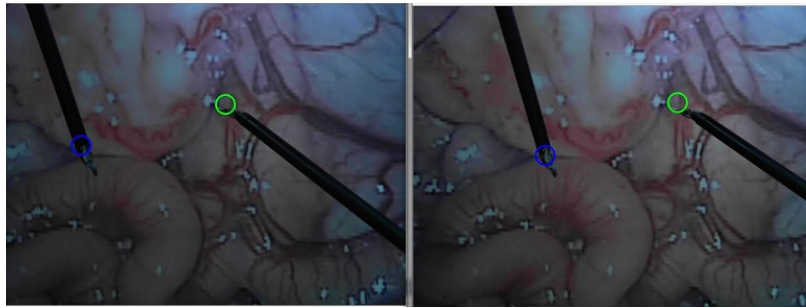


Figure 5.28: Arbitrary tool configuration 10 (Dark lighting conditions).

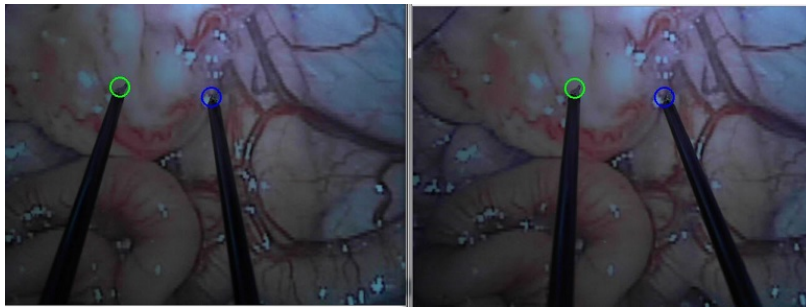


Figure 5.29: Arbitrary tool configuration 11 (Dark lighting conditions).

As indicated by Figures 5.28 and 5.29, the darkened lighting conditions do not have a negative effect on the tracking algorithm. The tool edges and therefore the tool tip are still detected even though the difference in pixel intensities between the tool shaft and the background has been decreased. This is attributable to the threshold on the edge detection function. This threshold is variable and therefore can be automatically adjusted to detect edges even for darker images.

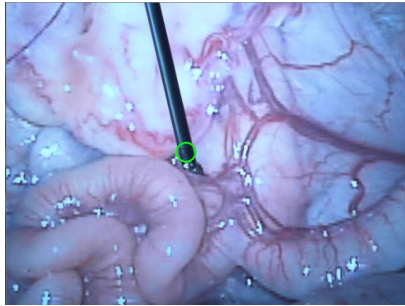


Figure 5.30: A Slight tool tip detection error.

The final issue found while tracking using the PHT algorithm is the instance of tool localization moving up the tool shaft (Fig. 5.30). This typically occurs when the edges of the tool shaft near the wrist are not clearly defined. Since the tool tip is detected wherever the end of the hough transform line is, the tool tip is subject to errors in the PHT algorithm. While this movement up the shaft is typically very small and infrequent, this type of error can still effect the localization. Fortunately, several safe-guards are implemented to mitigate the effects of this error. The pixel location from both channels is averaged before the Cartesian registration step is performed. Generally, this detection error is only present in one channel at a time and can therefore be corrected using an average of both channels location. This helps minimize the deviation in Cartesian space due to this detection error. Using the tool tip location from each channel, the disparity calculation is then subjected to a moving average filter. This filter helps constrain slight detection errors before the depth is calculated. While this slight detection error is acknowledged, its effects are actively minimized.

In reviewing the different tool, lighting and background conditions, several conditions were found where tools could and could not be tracked. For the majority of configurations tested, the tracking algorithm still located the tools according to manual inspection. The primary issues arose when the tool tip was not visible or the tool shafts occluded the tool tip. While this investigation was primarily a qualitative one, it appears the tracking algorithm can perform in less than ideal conditions. However, this analysis is for data taken from an experimental setup in a laboratory setting and therefore the results in terms of varied conditions may change for the endoscopic camera in real operating room conditions. Additionally, it should be noted that for different

‘zoom’ factors, the ideal separation of the tool shaft needs to be manually adjusted. This detracts from the autonomous nature of the tracking algorithm slightly, but will be fixed in future revisions.

5.4.1 Performance Analysis

Overall the performance of the experimental setup for the surgical tool tracking algorithm meets the majority of the design goals outlined in Section 3.1. In terms of computational speed, the algorithm takes an average of **39.9 ms** for each frame to be analyzed and the tool position calculated. This results in working framerate of around **25.86 FPS**. This working framerate is sufficient, considering the upper bound of this performance is limited to the 30 FPS with which the webcam can capture video. This speed is an improvement over previously reported frame rates of 15 - 17 Hz [35], [37], [45]. In addition, this framerate exceeds our design goal of 16 Hz framerate.

In terms of accuracy in tool tip localization, the reprojected tool position has a stationary noise level of **1.293 mm**. This noise level is primarily a result of the separate tool tracking channels. Despite this noise level the average reprojection error is found to be **3.05 mm**. This error is promising considering the area that the surgical tool tip covers is approximately $8mm^2$ in Cartesian space. The 95th percentile of tool rejections are within **5.474 mm** of the known location. The percentage of time where the surgical tool tip is located with 4 mm of the known trajectory is found to be **78.88%**. Similarly the percentage of time where the surgical tool is located within 8 mm of the known trajectory is **99.4%**. This percentage accuracy also exceeds our design goal of 86% percent accuracy (within 8 mm).

In relation to ‘robustness’ of the algorithm, the tool tracking algorithm appears to perform in sub-optimal conditions including varied background objects and poor lighting conditions. However, this evaluation was qualitative and therefore a more rigorous analysis will be required for the operating room setup.

Chapter 6

Implementation in da Vinci Surgical System

Given the successful implementation of the surgical tool tracking algorithm with an experimental camera setup (Chapter 4), the tracking algorithm was transitioned to use the da Vinci camera unit to integrate with the entire surgical system (Fig. 6.1). The da Vinci camera unit is a high definition stereo camera mount similar to the experimental camera setup. However the video feed and its processing is slightly more complicated. First the implementation and integration of the tracking algorithm into the da Vinci surgical system is presented in Section 6.1. Then the stereo calibration model for the surgical endoscope is presented in Section 6.2. With the tracking system integrated with and calibrated for with the da Vinci, performance data will be calculated for tracking with the da Vinci system (Sec. 6.4). Finally, tool tracking will be demonstrated and assessed on real operating room footage in Section 6.5



Figure 6.1: A mock up depiction of the tracking algorithm implemented as in an operating room.

6.1 Surgical Endoscope Integration and Implementation

In order to implement the tool tracking algorithm in conjunction with the da Vinci Endoscopic Video Camera, certain augmentations had to be made to the software environment as well as the hardware. Accessing the live video feed from the da Vinci Endoscopic camera required the use of a dual DVI to USB capture device as described in Section 3.7. The da Vinci video system utilizes two Panasonic © GP-US932A HD camera control units (Panasonic Corporation, Kadoma, Osaka, Japan). Each of the camera control units receives the video feed from one channel of the endoscope and passes the feed along to a monitor for viewing. In the case of the da Vinci S © video system, the camera control units output the video signal through an Serial Digital Interface (SDI) connector. Normally, the SDI signal is then used to send video to the master console for the surgeon. However, in order to extract this video in real time the SDI signal was first converted into HDMI so that it could be captured by the DVI to USB capture box. In order to convert the SDI to HDMI, a generic conversion box was used. The SDI to HDMI conversion introduces very little loss of quality. The HDMI can then easily adapted to a DVI signal for use with capture box.

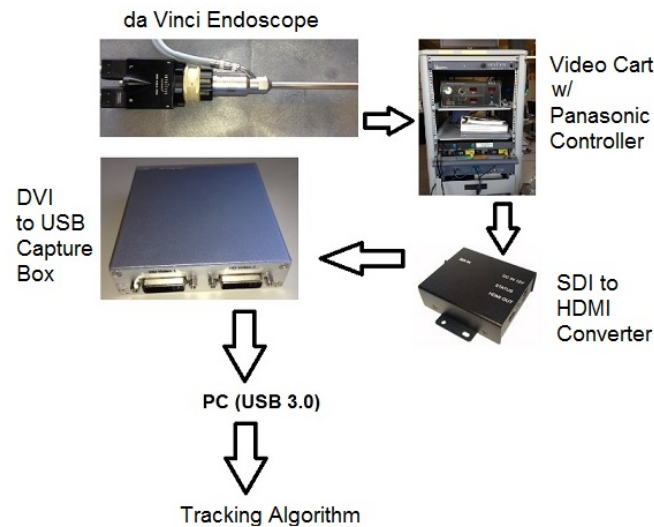


Figure 6.2: The sequence of video conversion and acquisition.

Once the video has been acquired and converted, the video feed can be analyzed by the OpenCV library in the same fashion as the experimental webcam setup. The video from the DVI capture box is managed by the capture thread and individual frames are extracted and added to the stereo channel buffers. The main difference in software for the capture thread is the resolution. While the da Vinci video cart captures frames at a resolution of 1920x1080 pixels (interlaced), this resolution is too high and would result in lengthy computational times. To avoid this the video frames are down sampled to a resolution of 720x480 in the capture thread of the program. This resolution is not at the same aspect ratio as the initial 1080p resolution. The additional vertical pixels are filled by black pixels on the top and bottom of each frame. These bars keep the ratio the same as the original video and can be ignored in terms of analysis. This 720x480 resolution is then used for all subsequent analysis in the program. For actual image analysis, the frame is cropped to include only 720x405 and thus the top and bottom black bars are removed.

Unfortunately, the conversion from SDI to HDMI when coupled with the DVI capture box reduces the capture frame rate significantly. While the resolution is down sampled in the capture thread, the SDI to HDMI converter and the DVI capture box both have to deal with the full 1920x1080 resolution in stereo. This high resolution

effectively reduces the capture frame rate to 10 FPS. While this poor frame rate does not effect saved video acquired by surgeons from the da Vinci video cart, it does affect the live video acquired for calibration and performance characterization.

It should be noted that while this video acquisition setup is necessary for compiling calibration and accuracy data, such a system would not be required for practical offline analysis. For surgeons to use this system after a surgery to analyze their tool movements, only saved videos from each stereo channels would be required. The next major alteration to the tool tracking application was to adjust the depth extraction and Cartesian registration portions of the tracking method. Since the camera alignment and lens configuration for the endoscope is clearly very different from the experimental webcam setup, a completely new calibration routine was required. It was assumed however, that the general model for the disparity - depth mapping would be similar.

6.2 Surgical Endoscope Calibration

In order to calibrate the endoscope stereo cameras for depth extraction, a similar calibration board to the one described in Section 4.1.1 was again utilized. The new calibration board was identical to the experimental setup board except for the scale (Fig. 6.3). The cross hatch marks on the second calibration board use more closely spaced hatch marks (16.15 mm) as opposed to the 24.5 mm spacing on the experimental board. The calibration board was then affixed on a plexi-glass sheet. The mount was then outfitted with a hole through the origin of the calibration board so that a mounting screw could be affixed to the calibration setup and simultaneously center the calibration board to the axis of the mounting board.

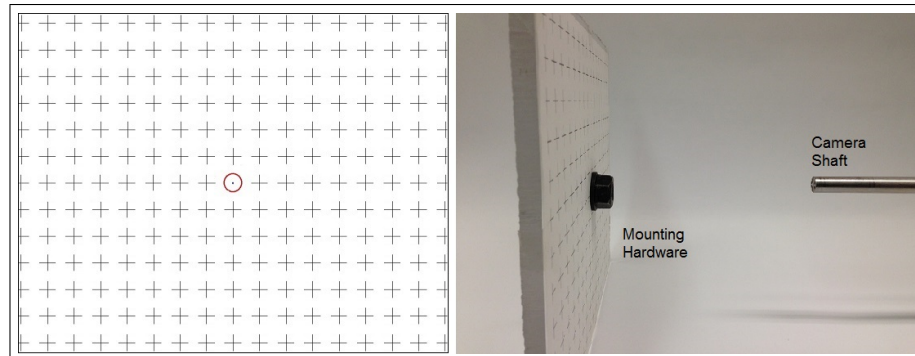


Figure 6.3: The Calibration board used for the endoscope (16.15 mm spacing).

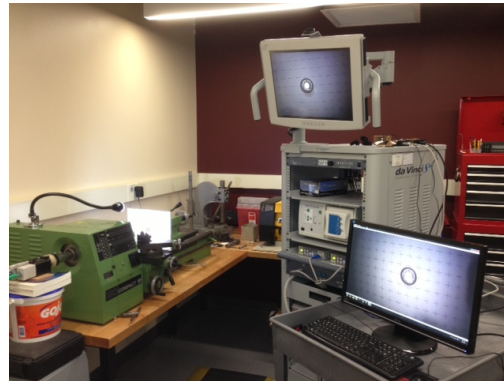


Figure 6.4: The setup used to acquire both calibration and performance data.

The calibration setup was designed to ensure a high level of accuracy. Since the calibration board requires precise known baseline depths when comparing disparities, the calibration was performed using a machinist's lathe (Fig. 6.5). The lathe ensures that the cylindrical object mounted in the spindle on one end and an object mounted in the tailstock on the other end are concentric. For the purpose of this calibration, the long cylindrical shaft of the endoscope was secured in the spindle (with the power off) and the clamp tightened around the scope shaft. The camera unit protruding from the opposite end of the headstock was also supported externally. The calibration board was then affixed in the tailstock end.

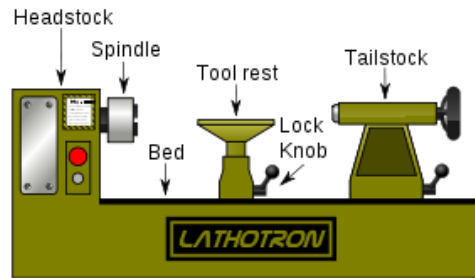


Figure 6.5: A standard machinist's lathe.

Once the camera and the calibration board have been positioned, the tailstock can be linearly translated at controlled displacements ($\pm 0.01''$ accuracy) in order to move the calibration board in known increments. In order to start the calibration procedure, the distance from the camera to the board was measured using a caliper. Using this 'rough' measurement as a baseline, all subsequent depths are adjusted using the linear stage. As with the experimental calibration for any particular depth, the calibration board is viewed through both stereo channels and all visible cross hatch marks are clicked in succession and registered in the calibration software in order to determine the disparity.



Figure 6.6: The endoscope calibration setup using a machinist's lathe.

The range of depths used for the calibration of this camera were between 150 and 250 mm. This range of depths was partially chosen so as to include depths at which the smallest of the trajectory board paths was visible. Using the video acquisition setup,

each calibration was analyzed and the disparities recorded for the span of pixel space. Once this data was acquired it was exported for offline analysis.

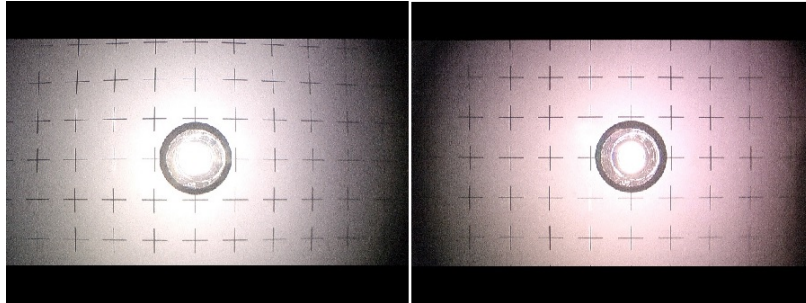


Figure 6.7: The left and right channels from the endoscope camera during calibration on the machinist's lathe.

Upon first inspection of the disparity vs. depth relationship, it became clear that the endoscope stereo setup was far different than the webcam configuration. The standard relationship between depth and disparity is that an object will have a smaller disparity as it is moved farther away. In the case of the endoscope cameras, the relationship is reversed so that objects that are closer have a slightly smaller disparity. This information implies that the cameras are situated with the optical axis diverging out from a parallel configuration (Fig. 6.8). As a result, the stereoscopic depth extraction method previously developed would not work.

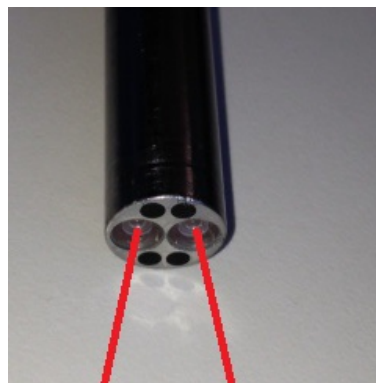


Figure 6.8: The lenses on the endoscope tip and the apparent optical path.

In addition to the non-parallel optical axis, the endoscopic cameras also include

a large amount of radial distortion or ‘fisheye’. The radial distortion adds an extra factor into the disparity calculation for the stereo cameras. At the extreme corners of the image, the disparity is magnified and therefore causes an error in the depths calculation. For the depth vs. disparity relationship, this radial distortion results in a large range of disparities at each calibration depth (Fig. 6.9).

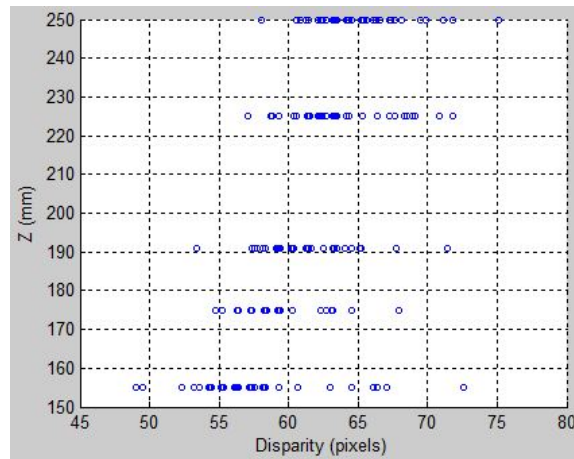


Figure 6.9: The depth (mm) vs disparity (pixel) for the endoscope calibration.

This large variance in disparities for a fixed distance is not sufficient for determining depth. Therefore the x and y pixel position (x_p, y_p) must be used to offset the range of disparities to a single disparity per depth. It is possible to implement a more sophisticated ‘fisheye’ correction filter, however the documentation concerning the da Vinci endoscope and camera setup is very minimal and provides no specific details on the endoscope optics, making such a correction filter difficult to design. In order to determine this offset mapping, a 3-dimensional polynomial was fitted to each data set.

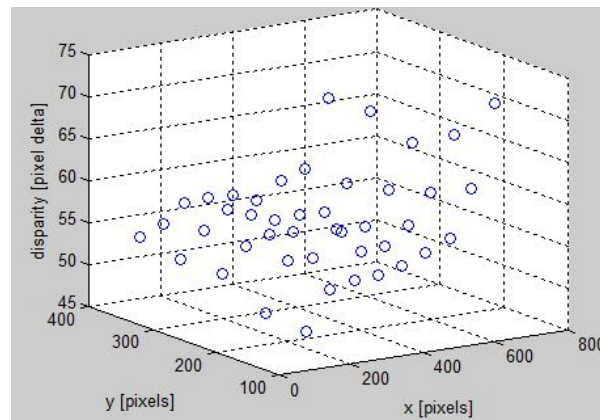


Figure 6.10: The disparity (pixel delta) is plotted as a function of x and y pixel location.

Since the disparity for each calibration depth must assume a single value. The offset due to the x and y pixel position (x_p, y_p) must be mapped to the offset: $disparity - disparity_{origin}$. The $disparity_{origin}$ value in this case was determined from the 8 disparities found in the center of the calibration board for each calibration depth. Using this offset the range of disparities were re-centered around the origin (Fig. 6.11).

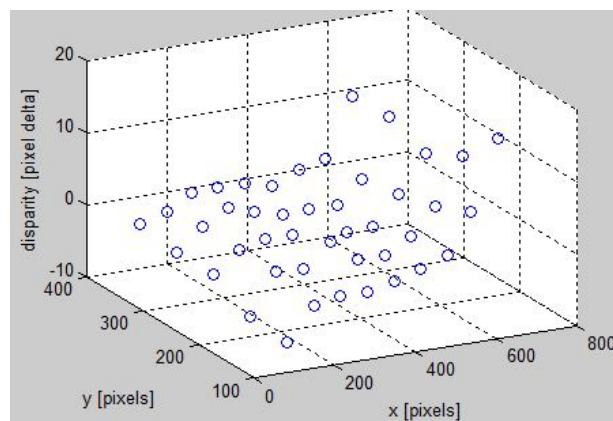


Figure 6.11: The disparity (pixel delta) offset re-centered and plotted as a function of x and y pixel location.

Using this deviation from the $disparity_{origin}$ value, we determined a 3 dimensional polynomial which could correctly describe the (x_p, y_p) radial distortion. To determine

this polynomial we utilized the curve fitting tool (cftool) from Matlab. While experimenting with different fit models, it became apparent that the optimal solution was a third order polynomial with cross terms. Other models including higher order polynomials were fitted to the data, however it was apparent that the third order polynomial was best correlated to the data. The third order polynomial took the form of Equation 6.1,

$$D(x, y) = p_{0,0} + p_{1,0}x + p_{0,1}y + p_{2,0}x^2 + p_{1,1}xy + p_{0,2}y^2 + p_{3,0}x^3 + p_{2,1}x^2y + p_{1,2}xy^2 + p_{0,3}y^3 \quad (6.1)$$

Where x and y represent pixel offsets from the center of the image and D represents disparity offset.

Using this polynomial, a plane was fit to the disparity data for all 5 baselines depths. The baseline depths tested were (155, 175, 191, 225, 250 mm). While the 250 mm baseline seemed to have more error due to pixelization, this data was still included.

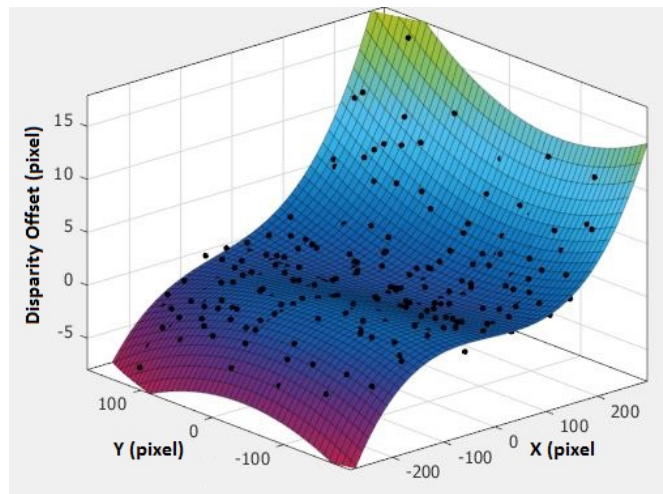


Figure 6.12: The disparity offset fit to a third order polynomial, a function of x and y pixel location.

This third order polynomial fit has the following coefficients:

$$p = \begin{bmatrix} p_{0,0} \\ p_{1,0} \\ p_{0,1} \\ p_{2,0} \\ p_{1,1} \\ p_{0,2} \\ p_{3,0} \\ p_{2,1} \\ p_{1,2} \\ p_{0,3} \end{bmatrix} = \begin{bmatrix} -0.2782 \\ -0.01216 \\ -0.002626 \\ 3.312e - 05 \\ 6.151e - 05 \\ 9.884e - 06 \\ 5.155e - 07 \\ 9.715e - 08 \\ 5.644e - 07 \\ 6.742e - 09 \end{bmatrix} \quad (6.2)$$

Where p represents the coefficient vector for the polynomial.

Using these coefficients to model the disparity offset we achieve a correlation coefficient of $R^2 = 0.9206$. This value is sufficient for our model given the potential for inaccurate pixel selection during calibration. For testing purposes, the fitting polynomial was raised to an order 5 with cross terms and the correlation only rose to $R^2 = 0.93$, implying that a more complex model did not produce a significant improvement in accuracy. Using the initial model to offset the disparity and account for the (x_p, y_p) radial distortion error, we achieve a more practical model for transforming pixel disparity into depth. After offsetting the disparity at each depth using this model, the disparity vs. depth relationship is again analyzed. While there still exists variance in disparity at each fixed depth, the range of this variance is limited using the offset term. The remaining variance is most likely due to noise in pixel selection during calibration.

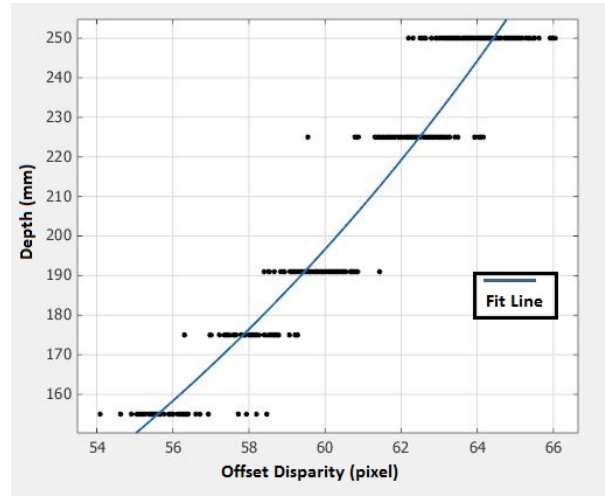


Figure 6.13: The depth vs. disparity relationship using the offset term for radial distortion.

This data was fitted to several different models but it was determined that an exponential model best suited the depth - disparity relationship (Fig. 6.12). In the case of the fit, the equation used was a single term exponential,

$$Z_w = A_0 e^{A_1 disp} \quad (6.3)$$

Where ‘disp’ is the offset disparity and A_i are coefficients.

Again, using the ‘cftool’ the coefficients for the depth disparity relationship were determined to be:

$$A = \begin{bmatrix} A_0 \\ A_1 \end{bmatrix} = \begin{bmatrix} 7.615 \\ 0.05419 \end{bmatrix} \quad (6.4)$$

Using the exponential equation 6.3 and the coefficients outlined in 6.4, the correlation is determined to be $R^2 = 0.92$. Using this model, we implement the Cartesian registration in the surgical tool tracking algorithm. As with the experimental camera calibration, given a sufficient model for depth extraction, the world (X_w, Y_w) coordinates can be calculated from the pixel coordinates (x_p, y_p) . The model used to calculate these coordinates has been found to be directly proportional to the depth. Therefore

the models for Cartesian registrations are

$$X_w = B_0(x_p Z_w) + B_1 \quad (6.5)$$

$$Y_w = C_0(y_p Z_w) + C_1 \quad (6.6)$$

Again using the fitting tool in Matlab, we determine the best fit coefficients for this model to be:

$$B = \begin{bmatrix} B_0 \\ B_1 \end{bmatrix} = \begin{bmatrix} 0.00155 \\ -2.495 \end{bmatrix} \quad (6.7)$$

$$C = \begin{bmatrix} C_0 \\ C_1 \end{bmatrix} = \begin{bmatrix} 0.001547 \\ 2.674 \end{bmatrix} \quad (6.8)$$

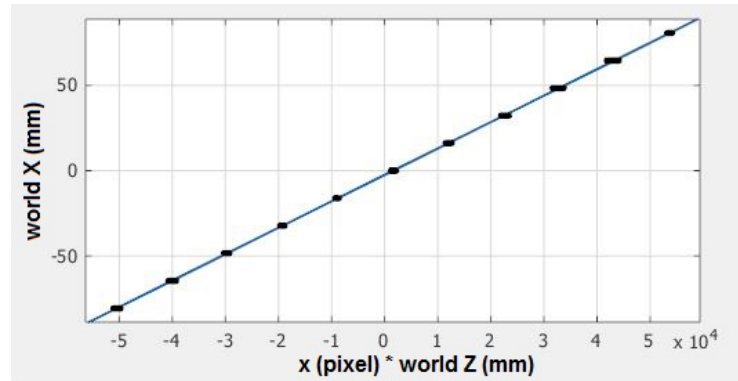


Figure 6.14: The relationship between X Pixel * World Z vs. world X.

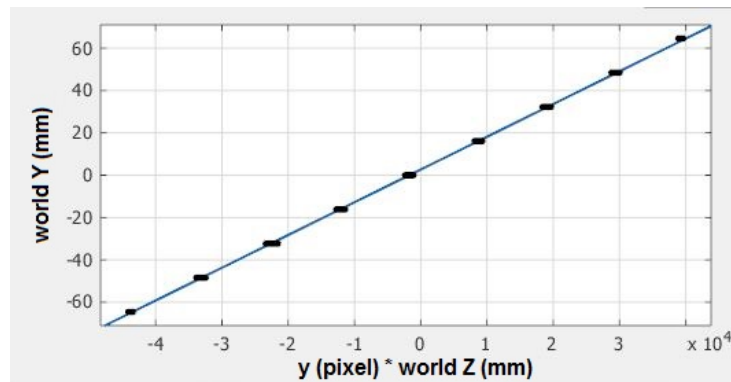


Figure 6.15: The relationship between Y Pixel * World Z vs. world Y.

The model for X_w (Eq. 6.7) yields an accuracy of $R^2 = 0.9998$ and the model for Y_w (Eq. 6.8) has an accuracy of $R^2 = 0.9996$. Both of these models have a better fit than the depth - disparity model because they have a higher signal to noise ratio and therefore avoid the high error levels from the disparity (Fig. 6.14, 6.15). Using the depth extraction equation (Eq. 6.1, 6.3) and the Cartesian registration equations (Eq. 6.5, 6.6), we then reproject the calibration data points from the pixel information and compare them with the known grid locations.

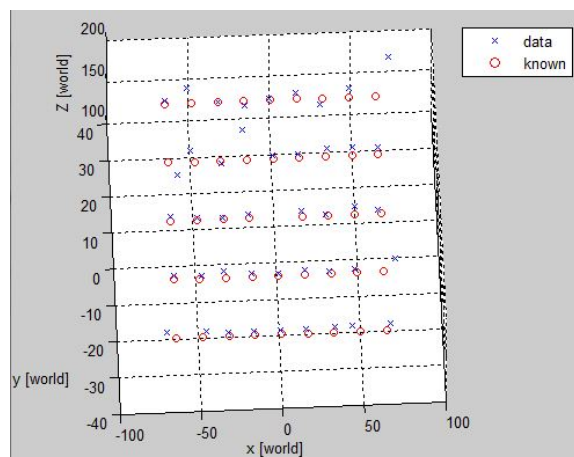


Figure 6.16: A top view of the calibration board reprojection (150 mm).

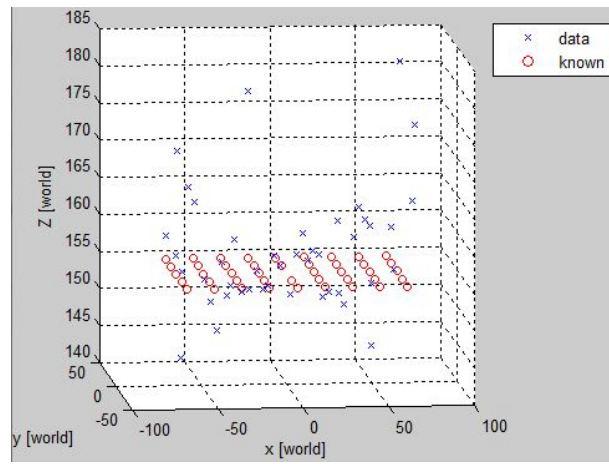


Figure 6.17: A side view of the calibration board reprojection(150 mm).

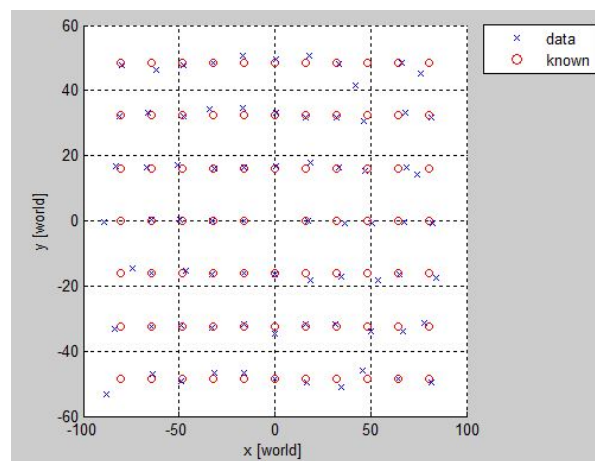


Figure 6.18: A top view of the calibration board reprojection (225 mm).

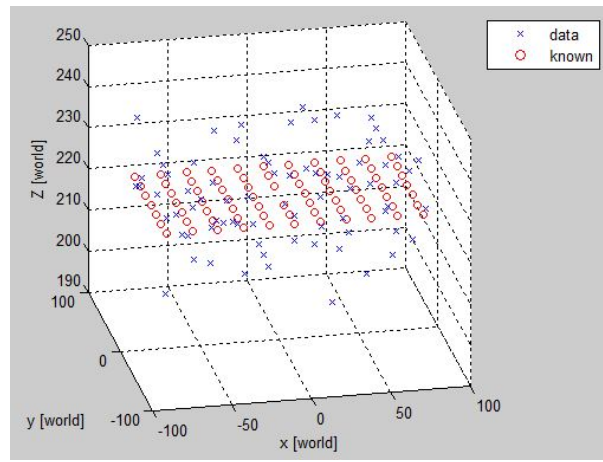


Figure 6.19: A side view of the calibration board reprojection (225 mm).

Figures 6.16 and 6.18 indicate a successful Cartesian registration in the (X_w, Y_w) plane. The model used for the reprojection in this plane achieves an average error of **2.225 mm** (Std Dev = 2.0517). This is relatively good in comparison with the Z_w data. As indicated by Figures 6.17 and 6.19, the Z component of the error in Cartesian coordinates is very significant. An average reprojection error of **7.885 mm** (Std Dev = 6.46) is attributable to calculation of the Z component. We attribute this poor accuracy in the depth extraction to the minimal range of disparity fluctuations compared with a large range of baseline depths (Fig. 6.13). This calculation is further confounded by slight errors in the radial distortion offset model (Fig. 6.12). While this depth extraction error is not ideal, the model is limited by the physical constraints of the endoscope camera. The most detrimental of these physical constraints is the 5.1 mm interocular separation. This severely limits the range of disparities found for the given work volume. Moreover, the lack of information about the internal optics precludes more sophisticated mathematical models to account for deviations.

Although the reprojection errors are less than ideal, the 7.885 mm reprojection error from the Z_w component is still smaller than the width of the wrist tracking area. Furthermore the (X_w, Y_w) Cartesian registration error (2.225 mm) is much smaller than the tracking area. Therefore, the tool tracking program was fully implemented using the depth extraction and Cartesian registration models for the endoscope camera setup and resulted in tracking to within the width of the tool.

6.3 Surgical Endoscope Performance Assessment Setup

In order to quantify the performance of the tool tracking algorithm using the surgical endoscope, experimental setups were implemented. Each experiment setup is very similar to those described in Chapter 5 and therefore each setup is briefly reviewed here. All performance data was collected using the computer system described in Section 3.7. Using the video acquisition setup from the depth calibration (Sec. 6.1), the video stream from the surgical endoscope was captured by the tracking program for analysis. For all experiments, the endoscope was aligned to the trajectory board origin using the lathe in order to precisely control camera angle and depth of field. Three primary metrics were quantified for the endoscopic setup: computational time, reprojection noise, and tracking accuracy.

In order to determine the computational time for the endoscopic setup, an identical set of experiments was performed to those described in Section 5.2. This experiment involved moving tool shafts through the visible field and tracking its location. Meanwhile the computational time for each thread in the program was recorded and sent to a data log. The three key tracking components are the capture thread, 2D detection and 3D localization. While the capture thread for this setup is required to perform the additional step of down sampling and region extracting from each frame, this still does not add any considerable time to the tracking algorithm (< 0.02 ms).

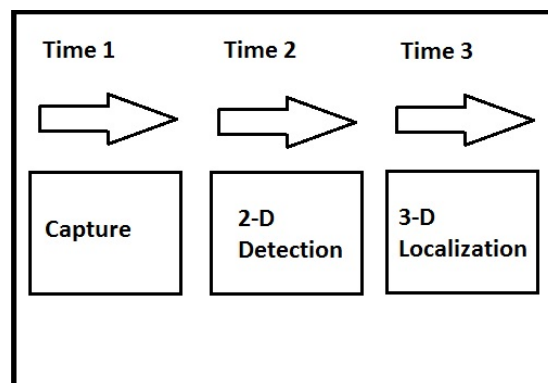


Figure 6.20: The timing component diagram from Section 5.2.

The computational time experiments were performed at a variety of depths and tool

configurations in order to assess the performance in a variety of conditions. After the time for each component of the program was assessed, the times were compiled into a total time according to Equation 6.9. Using this total time the frame rate (FPS) was subsequently computed.

$$T_{tracking}(ms) = Max(T_{detect,right}, T_{detect,left}) + T_{3DLocate} \quad (6.9)$$

In order to determine the noise associated with tracking a stationary tool, the same routine was performed as was used for the experimental webcam setup. A standard da Vinci forcep tool is held stationary in a clamp with the end effector in the field of view. The end effector is held still while the tracking algorithm is run for varied amounts of time. The tool location is then saved to a data log for offline analysis. In order to determine the amount of noise, the tool locations are then compared with an average center location. The error from each reprojected location is calculated as a deviation from this average location.



Figure 6.21: The tool tip is held still while the tracking algorithm is performed for noise characterization.

The final metric used to quantify performance is the accuracy of the tracking algorithm when compared with a known trajectory. This experiment was performed in a similar manner as for the experimental camera setup (Sec. 5.3). Using the video acquisition system, the known tracking board (Fig. 6.22) is aligned in the lathe just as the calibration board. The center of the tracking board is precisely oriented to sit inside the tailstock of the lathe using the same jig as the calibration board. Using this jig, the tracking board was adjusted to various known depths from the endoscope tip.



Figure 6.22: The trajectory board positioned in the lathe jig.

Several trial experiments were performed with the tool manually articulated through the known trajectory at various depths. These trials were used to determine the accuracy of the tool localization routine when compared with the known locations of the tracking path (Fig. 5.9). Due to the constrained field of view with the endoscope, only the smallest of the three paths on the trajectory board could be used. These model points were produced in Matlab for analytical comparison as a array of finitely spaced locations with 0.05 mm. The reprojected Cartesian locations of the tool tip are calculated in the tool tracking program and exported to a data log file for offline analysis. The reprojected error was computed using the standard distance formula.

6.4 Surgical Endoscope Performance Analysis

6.4.1 Surgical Endoscope Computational Speed Performance

The first performance metric to be analyzed is the computation speed. Using the setup explained in Sections 5.2 and 6.3, the computational speed performance was analyzed for the surgical endoscope setup. The timing data was taken after the tracking program had been altered to utilize the depth extraction and Cartesian registration models developed in Section 6.2. Timing data was taken while the tool tips were tracked in a variety of arbitrary configurations. In all, 14 different tool tracking trials were performed while timing data was collected.

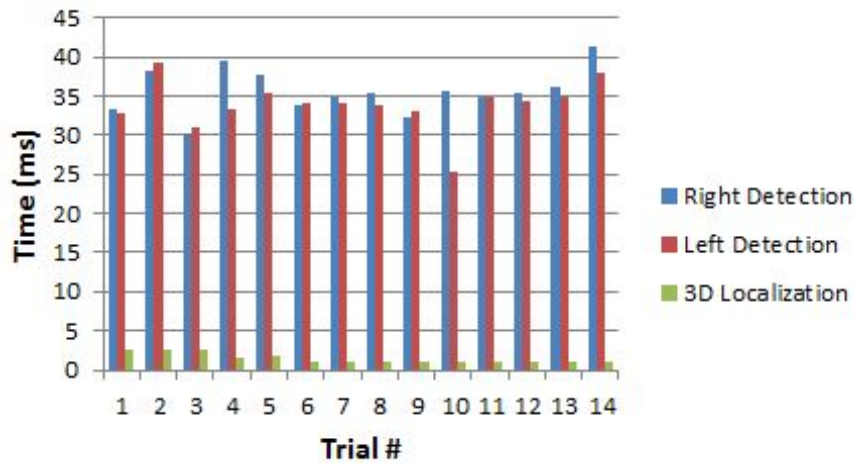


Figure 6.23: The various tracking components and their computational time (ms) over 14 repeated trial data collections of arbitrary tool motions.

As seen in Figure 6.23 the computational time for the left and right detection threads are around 35 ms each. The 3D localization thread takes approximately 1-2 ms. The longer of the two tracking threads and the 3D localization thread times are summed to calculate the total tracking time.

The total tracking algorithm time is used to compute the frames per second. While the live video acquisition system used with the endoscope is limited to around 10 FPS, this issue is attributable to an external hardware problem which can easily be addressed with a better converter. The software computation frame rate is the speed this work is concerned with characterizing. The computational frame rate for 14 different trials is given in Figure 6.24.

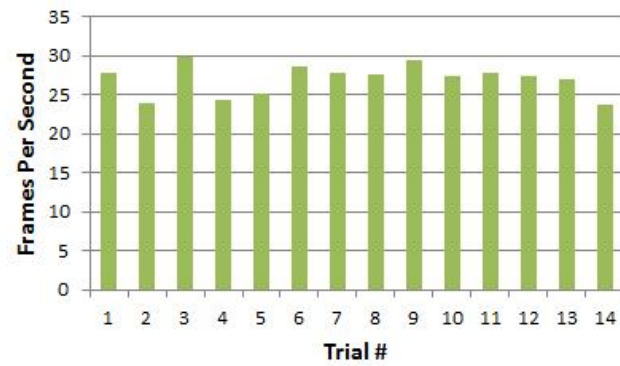


Figure 6.24: Frames per seconds for the endoscope setup using the optimized tracking algorithm, taken over 14 trial data collections of arbitrary tool motions.

Table 6.1: The time in ms is given for the left and right detection threads and the 3D localization thread. Using these thread times, the FPS is computed for the endoscopic video setup.

Trial #	Detection Right (ms)	Detection Left (ms)	3D Locate (ms)	Frames Per Second (FPS)
1	33.215	32.926	2.5813	27.86
2	38.2465	39.3472	2.496	23.898
3	30.359	30.906	2.6838	29.771
4	39.526	33.358	1.687	24.2634
5	37.775	35.518	1.8972	25.2067
6	33.8077	34.0275	1	28.549
7	34.9524	34.039	0.9913	27.8213
8	35.2917	33.7448	0.9635	27.5822
9	32.2775	32.9948	0.9372	29.4713
10	35.5782	25.2891	0.9694	27.3616
11	34.994	35.003	1	27.7755
12	35.365	34.4489	1	27.499
13	36.0827	34.8722	1	26.9667
14	41.242	37.983	1	23.6729

The computational speed performance for the endoscopic video setup is just as fast if not slightly faster than the experimental webcam setup. The average 2D object detection time for the right channel is **35.62 ms** while the detection time for the left channel is **33.89 ms**. The average 3D localization thread time is **1.44 ms**, this is very close to 3D localization time for the experimental webcam setup which implies the depth extraction and Cartesian registration model for the endoscope does not require a significantly longer computation time. These individual values result in an overall computation time of 37.26 ms. This overall computation time yields an average working frame rate of **26.98 FPS**. While this obviously does not take into account the live video frame rate restriction of 10 FPS, this frame rate would be attainable given a faster video acquisition system. Since we have no reason to believe the endoscopic video frames can be analyzed faster than the experimental webcam frames, it can be safely assumed that the improved average frame rate for this setup can be attributed to a slight increase in CPU performance on the day this data was acquired.

6.4.2 Surgical Endoscope Tracking Noise

Using the endoscopic camera setup and video acquisition described in Section 6.3, the tracking noise for a stationary tool was quantified. Tool location data was collected for a variety of depths. The tool depth was varied using the lathe jig between 50 and 300 mm. The tool tip location in the field of view was varied so that data could be analyzed in the whole pixel space. In order to visualize the noise experienced by the tracking algorithm, the tool location rejections are plotted against the average location point for the stationary tool.

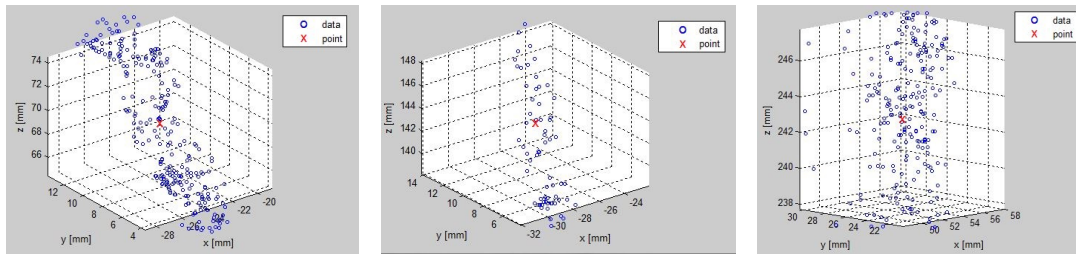


Figure 6.25: The reprojected tool tip locations for a stationary tool at various heights: 70, 143, and 242 mm. (Using the da Vinci endoscopic camera).

Upon inspection of Figure 6.25, it is apparent that the depth extraction and Cartesian registration model for the endoscopic camera results in a greater amount of tracking noise than the webcam setup. For the stationary tool, the tracking noise experienced is as much as 5 mm. The figures clearly indicate that this noise is experienced primarily in the Z_w component of the position. The various noise values are presented in Table 6.2.

Table 6.2: The reprojected coordinates for the stationary tool are compared with the average positions in order to determine the noise (For the da Vinci endoscopic camera).

Trial #	Depth Z (mm)	X Deviation (mm)	Y Deviation (mm)	Z Deviation (mm)	Overall Deviation (mm)
1	242.7054	2.8388	1.2755	11.0286	11.669
2	69.4828	2.258	0.7969	6.187	6.819
3	143.171	1.2636	1.0161	8.609	8.9231
4	151.7156	2.4668	1.002	9.1046	9.6565
5	115.7346	1.5916	1.1709	4.802	5.4512
6	188.2524	0.9068	0.5011	4.7917	4.9956

The calculated noise for a stationary tool when compared with the average location is found to vary anywhere between 4 and 11 mm. The average noise is found to be **7.92 mm**. When compared with the experimental webcam setup noise (1.29 mm), this noise characterization is very poor. However, it is important to note that the average noise in the Z component is found to be 7.42 mm. When compared with the average noise in the x and y components (1.89 and 0.96 mm, respectively) it is clear that the depth extraction component is the main issue. However, this noise is not in itself disappointing, considering the average reprojection error of 7.88 mm from the depth calibration (Sec. 6.2). It appears that a more effective depth extraction model may improve this noise.

6.4.3 Surgical Endoscope Tracking Accuracy

The final performance metric to be computed for the tool tracking algorithm using the surgical endoscope camera is the accuracy of the tool tip localization in 3D Cartesian coordinates. The tool tracking error is calculated as a deviation from a known trajectory board as described in Sections 5.3 and 6.3. The known trajectory board was adjusted to several different camera depths in order to observe the error for a variety of distances. Given the constraints of the endoscopic camera, the only path that could be articulated by the tool and stay in the field of view was the smallest of the three paths. Several data sets and the corresponding model data are given below in order to demonstrate the performance of the tracking algorithm. The measured data points are given in blue (x) and the reference trajectory points in red (o).

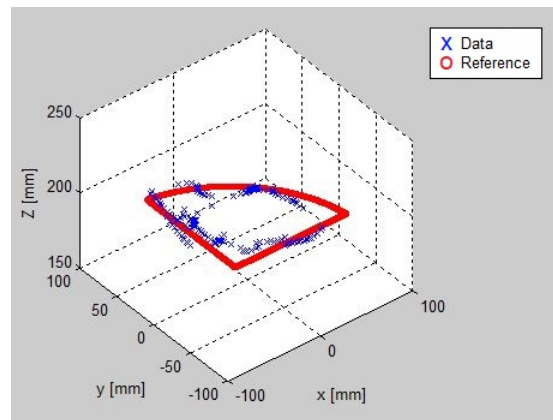


Figure 6.26: Tool motion reprojection for arc path 1 (198 mm).

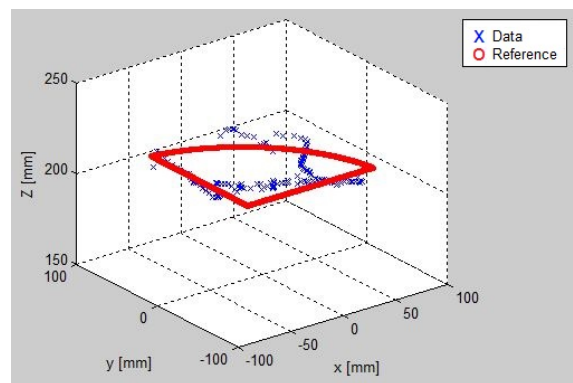


Figure 6.27: Tool motion reprojection for arc path 1 (212 mm).

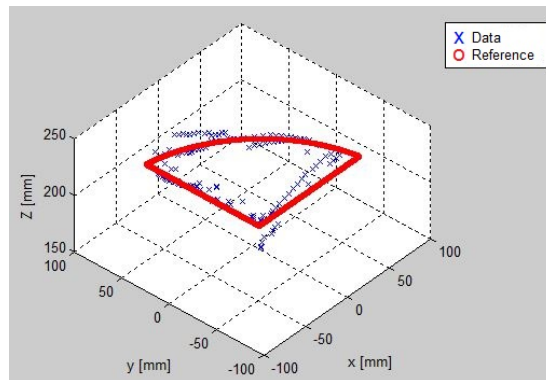


Figure 6.28: Tool motion reprojection for arc path 1 (225 mm).

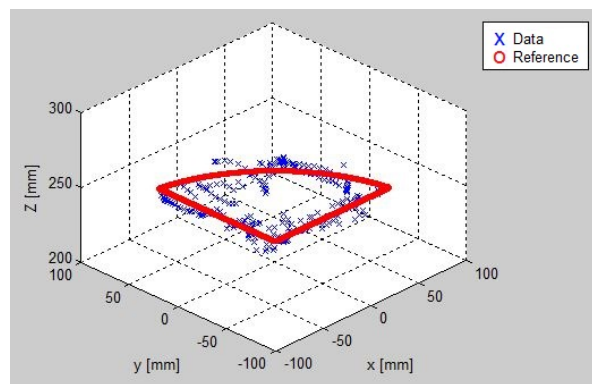


Figure 6.29: Tool motion reprojection for arc path 1 (248 mm).

Manual inspection of Figures 6.26 - 6.29, reveals that the majority of the reprojection error is a result of the depth extraction. The (X_w, Y_w) coordinates are generally more correct when compared with the Z_w reprojection. The error in depth extraction Z_w is again due to small jumps in the disparity between the tool tips in each channel. Given the depth extraction model for the endoscope, a single jump in pixel location can cause a change of up to 6 mm in depth. Theoretically, this would improve with full 1080p resolution.

Table 6.3 contains the set of trial tracking data and the computed reprojection errors. Each trial consisted of manually articulated the tool tip through the same reference trajectory. This articulation was repeated at multiple depths which were set

using the machinist's lathe. The table contain the baseline depth, the average depth in the reprojected coordinates, and the percent of rejections where the tool location was within an 8 mm range of the known trajectory. Tables 6.4 - 6.6 contain the same error metrics for the known trajectory tracking but instead contain the X_w , Y_w and Z_w error information. The baseline depths used for the accuracy analysis were between 190 and 280 mm. All trials were performed using the smallest arc trajectory.

Table 6.3: The reprojected coordinates are compared with the known trajectory coordinates in order to determine errors over 7 repeated traces of the same reference trajectory (Endoscopic camera setup).

Trial #	Baseline Z (mm)	Average Z (mm)	Avg Error (mm)	Percent in 8 mm	95 th Percentile Error (mm)
1	214	212.12	7.9413	51.97	14.4254
2	200	198.5	6.3821	70.7006	11.8231
3	245	245.9	16.7605	30.4965	21.043
4	227	225.46	6.2384	67.6259	14.0754
5	250	248.85	8.0946	54.1667	15.5525
6	252	251.03	8.608	46.9136	15.7002
7	286	285.69	6.7536	64.6341	12.821

Table 6.4: The reprojected X coordinates are compared with the known trajectory X coordinates in order to determine errors over 7 repeated traces of the same reference trajectory (Endoscopic camera setup).

Trial #	Baseline Z (mm)	Average Z (mm)	X Avg Error (mm)	Percent in 8 mm	95 th Percentile X Error (mm)
1	214	212.12	0.9552	98.0263	5.0502
2	200	198.5	0.2973	100	3.0152
3	245	245.9	0.3386	99.2908	1.6246
4	227	225.46	0.0193	100	0.0237
5	250	248.85	0.2538	100	2.7027
6	252	251.03	0.077	100	0.0246
7	286	285.69	0.5396	100	3.8477

Table 6.5: The reprojected Y coordinates are compared with the known trajectory Y coordinates in order to determine errors over 7 repeated traces of the same reference trajectory (Endoscopic camera setup).

Trial #	Baseline Z (mm)	Average Z (mm)	Y Avg Error (mm)	Percent in 8 mm	95 th Percentile Y Error (mm)
1	214	212.12	0.0105	100	0.0222
2	200	198.5	0.1355	100	0.7856
3	245	245.9	8.0814	84.3972	15.0999
4	227	225.46	0.0792	100	0.0233
5	250	248.85	0.0775	100	0.0236
6	252	251.03	0.2285	100	2.1377
7	286	285.69	0.1579	100	1.0701

The reprojection errors in the X,Y and Z components are reported in Tables 6.4, 6.5, and 6.6. Upon inspection it is clear that the error in the Z_w direction is the largest component of the error. After compiling the reprojection errors for the trial runs, the overall average error was found to be **8.68 mm**. The minimum total error found was **6.24 mm**. The maximum total error was **16.76 mm**. In comparison, the average reprojection error in the X_w direction was **0.35 mm** and the average reprojection error in the Y_w direction was **1.25 mm**. Finally, the average reprojection error in the Z_w direction was **5.16 mm**.

Table 6.6: The reprojected Z coordinates are compared with the known trajectory Z coordinates in order to determine errors over 7 repeated traces of the same reference trajectory (Endoscopic camera setup).

Trial #	Baseline Z (mm)	Average Z (mm)	Z Avg Error (mm)	Percent in 8 mm	95 th Percentile Z Error (mm)
1	214	212.12	5.3157	85.5263	12.8906
2	200	198.5	4.8895	78.9809	10.7664
3	245	245.9	7.058	63.8298	15.1106
4	227	225.46	3.8742	89.2086	12.9873
5	250	248.85	4.4971	87.5	9.5841
6	252	251.03	5.4315	75.9259	11.0508
7	286	285.69	5.0302	78.0488	12.713

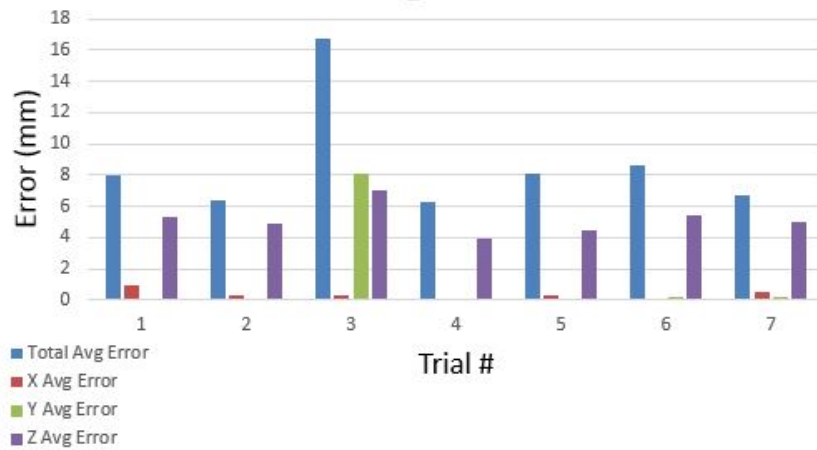


Figure 6.30: The average error (mm) in tool repositioning in the X,Y,Z components as well as overall errors.

The 95th percentile of total repositioning errors overall was **15.063 mm**. In comparison, the 95th percentile of errors in the X_w direction was **2.33 mm** and **2.74 mm** in the Y_w direction. This is in contrast to the 95th percentile of errors in the Z_w direction which was found to be **12.16 mm**.

The final metric calculated is the percentage of the time the tool is accurately located within certain ranges of the known trajectory. The overall reprojected location is accurate to within 2 mm of the known trajectory 8.68 % of the time. The reprojections were within 4 mm of the known trajectory 18.76% of the time. The reprojections were within 6 mm 38.806% and within **8 mm** of the known trajectory **55.22 %** of the time. In comparison the X coordinate of the reprojection is within 8 mm of the known locations **99.62%** of the and within 8 mm of the Y coordinate location **97.77%** of the time. Therefore it is obvious that the majority of the tracking error is due to the depth extraction data. The Z coordinate is within 8 mm of the known trajectory only **9.8%** of the time. However, even with the poor depth extraction data, the (X_w, Y_w) reprojection data is still very accurate.

6.5 Tracking Performance Using Real Operating Room Footage

A final test of the tracking algorithm was performed on a recorded video from a surgical procedure *in vivo* using the da Vinci robot. The specific video used was taken from a prostate removal procedure performed at the University of Minnesota medical school. For this analysis, a video containing both the stereo channels from a da Vinci surgery were used (Fig. 6.31). In order to gauge the robustness and efficacy of the tool tracking algorithm when used on a pre-recorded surgical video, the tracked tool tip position was inspected for successful tracking. The tool tip locations were only analyzed in terms of pixel space since the reprojected coordinates could not be compared with any known reference trajectory since there are not possible to establish within a patient. It is worthwhile to note that in contrast to the experimental webcam setup, a typical surgeon keeps the endoscope very close to the surgical tool tips. For this reason the ideal tool edge separation criteria in the tracking algorithm was changed to 60 pixels as opposed to the previous value of 25 pixels.

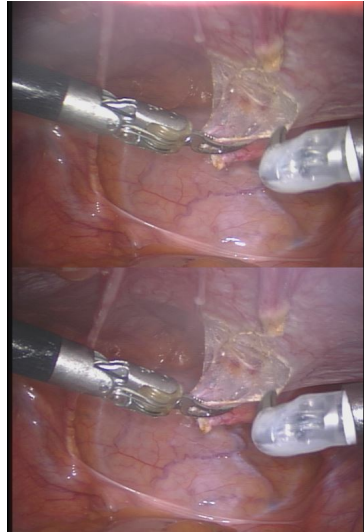


Figure 6.31: A stereo video frame from the sample prostate removal procedure.

For this evaluation the videos used were the left and right stereo channels from the endoscope, recorded at 640x480 resolution. In order to have a manageable video for manual inspection, a 54 second video clip was extracted from the longer procedure recording. This video clip was cropped to include only frames where the right and left tool tips were both visible. The sequence utilized also included smoke from electrocautery tools in order to test the robustness of the tracking algorithm in typical operating room conditions.

The saved procedure video was subjected to the tool tracking algorithm and the detected tool location was superimposed over each frame. Each stereo frame along with the detected edges were then recorded for offline manual analysis (Fig. 6.32). Using this recorded video, the superimposed tool location was manually inspected. Each frame was analyzed to determine if one, two or zero tools were successfully located. A successful location was defined as being within the confines of the wrist of the tool shaft.

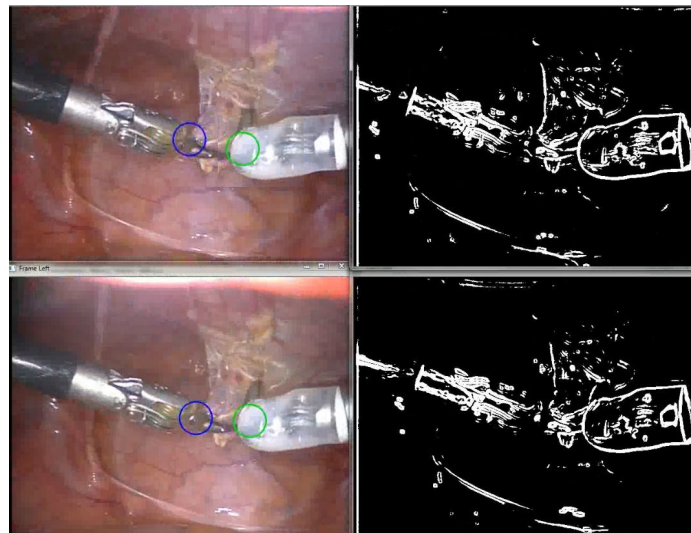


Figure 6.32: A single frame of the detected tool location from surgical procedure video.

Using the 54 second video clip of the surgical procedure, a total of 784 frames were analyzed. Of these frames, **11.98 %** contained zero correctly detected tool tips. In contrast, **88.01 %** of the frames contained at least one correctly located tool and **51.14 %** of the frames contained successful detections of both tool tips.

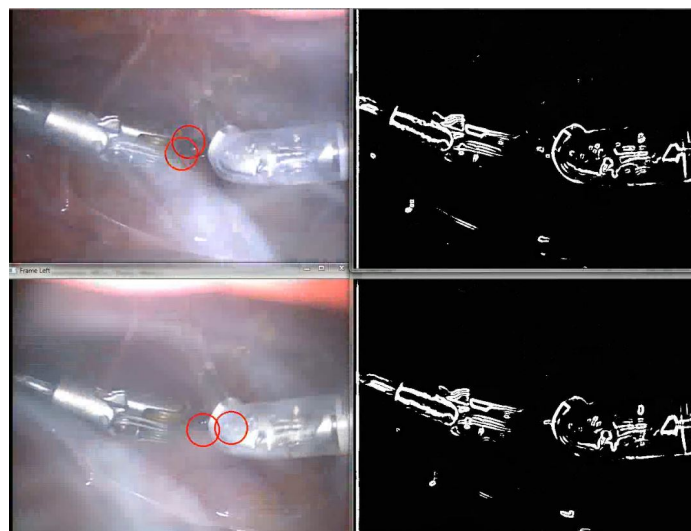


Figure 6.33: In the presence of electrocautery smoke, the tool tip cannot be found.

While the large amount of frames where neither tool tip was detected may seem high, these frames were largely the result of the electrocautery smoke found in the field of view (Fig. 6.33). The occlusion do this smoke was present for approximately 8 % of the frames. The presence of this texture in the foreground caused the outside edges of the tool to be lost. When the tool shaft edges cannot be found, this algorithm cannot find the tool tip. However, it has been acknowledged that one of the shortcomings of all computer vision tracking approaches is the line of sight requirement. However, the **88.01 %** of frames where at least one tool was detected is very promising.

6.6 Surgical Endoscope Performance Conclusion

Based on the performance data presented in Section 6.4, it is apparent that the endoscopic setup performs just as well as the experimental setup, if not better in terms of computational speed. The average frame rate for the surgical endoscope tracking was found to be **26.98 FPS**. This is approximately the same working frame rate as the experimental webcam setup (25.86 FPS). This frame rate is also a marked improvement over previously reported frame rates of 15 - 17 Hz [35], [37], [45]. Moreover, it reports tracking location in 3D Cartesian space, unlike most of the previous work.

It terms of tracking noise, the endoscope setup suffers from a high level of noise in the depth extraction model. For a stationary tool, the tracking algorithm returns an average noise value of **7.92 mm** in the tool reprojection data. When compared with the average reprojection noise in the experimental webcam setup (1.293 mm), this result is slightly disappointing. However, this noise can be attributed to the depth extraction model, which has an average model reprojection error of **7.885 mm**. This poor model correlation is a result of the constrained disparity variation inherent in a stereo camera with such a narrow interocular separation as well as the uncertainties in camera optics. The range of disparities found in this camera setup was found to be about 10 pixels for a depth range of 200 mm. This correlation results in a high signal to noise ratio. This suggests using higher resolutions (1080p) may result in better performance, however, this improvement would still be limited due to depth extraction issues.

It terms of tracking accuracy, the overall reprojection error when compared with the known trajectory was found to be **8.68 mm** while the 95th percentile of all tool

reprojection errors was **15.1 mm**. Similarly, the amount of reprojections within 8 mm of the known trajectory was found to be **55.2 %**. This tracking accuracy is not as promising as the accuracy for the experimental webcam setup. The webcam setup achieved an average reprojection error of 3.05 mm. However, it is important to note that the average reprojection errors for the endoscope setup in the X_w and Y_w components was found to only be **0.35 mm** and **1.25 mm**, respectively. In comparison the average reprojection error in the Z_w component was **5.15 mm**. The majority of the tracking error is due to the depth extraction model.

Finally, the robustness of the tracking algorithm was evaluated for a recorded surgical procedure. The superimposed tool tip locations in pixel space were manually inspected to determine if the tool tip was accurately located. For the 54 second video clip used, both tool tips were correctly located **51.2 %** of the time. Similarly, at least one tool tip was correctly located **88.0 %** of the time. While neither tool tip was correctly located **12.0 %** of the time, this error is primarily due to occlusions from electrocautery smoke. In comparison, the method presented by Reiter et al. was also subjected to a manual frame-by-frame analysis. This group reported a 93 % accuracy over 1600 test frames [47]. While this approach achieves a higher percentage of successfully located frames, it does so at a much slower frame rate (1.2 secs/frame) and provides no Cartesian location data. In part, this difference is due to the fact that Reiter et al. focus exclusively on wrist features while our approach focuses on the tool shaft. Since wrists are in the field of view more often than a tool shaft, it is reasonable to expect better tracking percentages. However, given the computational advantages of this work, it may be beneficial to employ our approach when the tool shaft is visible and Reiter's when only the wrist is visible. This could leverage the unique benefits of both approaches.

In terms of design goals, the tool tracking algorithm when performed using the endoscopic camera unit still meets a majority of the requirements. The speed and timing data goals are still met with excellent results. The speed goal of 16 Hz is more than met with the 26.98 FPS frame rate and related timing data output. The tool tracking algorithm also remains agnostic to the tool type used. Additionally, the endoscopic setup does not change the algorithms simplicity or ease of use. The program can still be distributed with the proper libraries and be run on a standard PC. In terms of accuracy, the tracking algorithm does not quite meet the 86 % localization requirement. Using

the endoscopic setup (emulating a real surgical procedure) the tracking algorithm can locate the tool tip within 8 mm 55.2 % of time. However, it is reasonable to expect that a more accurate depth extraction method could improve this tracking accuracy to a sufficient level particularly since the X_w and Y_w tracking components provide much higher accuracies. Finally, design goal of robust tracking is ambiguous. The ability of this algorithm to detect the tool tip in the face of changing lighting conditions and specular reflectance is just as favorable as in the webcam setup. However, upon investigation with the sample surgical procedure videos, the presence of complicating factors like electrocautery smoke or only partial views prove detrimental to the tracking algorithm. However, the presence of such occlusions is known to affect the performance of all computer-vision based methods and this was acknowledged in the design criteria.

Chapter 7

Conclusion and Discussion

7.1 Overview Of The Presented Methodology

The widespread use of surgical robotic procedures in the past decade has had positive effects related to patient recovery time and reduced blood loss. The da Vinci surgical system from Intuitive (Intuitive Surgical Inc, Sunnyvale, CA) is currently the only general surgery robot approved for use by the Food and Drug administration (FDA). While this surgical system has had many positive benefits, there are still issues related to training and certification when using such a system. Several tool motion metrics have been determined to provide quantitative feedback relating to the skill level of the surgeon. These metrics include path length, jitter, motion economy and completion time. In order to develop a training routine capable of reporting these metrics, a surgical tool tracking algorithm is required. Currently, no low-cost tool tracking algorithm exists which can locate the absolute position of the tool in real time with acceptable accuracy.

In this work, we present a novel, computer-vision based approach for the tracking of surgical robotic tools. The method presented exploits certain known geometric constraints of the surgical tool shaft in order to locate the tool (Fig. 7.1). Specifically, we use the probabilistic Hough transform to quickly identify the lines which outline the surgical tool shaft. Once these lines have been identified, the separation between the two lines and the extent to which the lines are parallel are used as metrics to determine which lines are the those corresponding to the tool shaft. While other methods were also examined for efficacy (depth segmentation and Haar classifiers), the joint Hough

transform —geometric constraint approach proved to be more reliable and accurate in detecting the surgical tool.

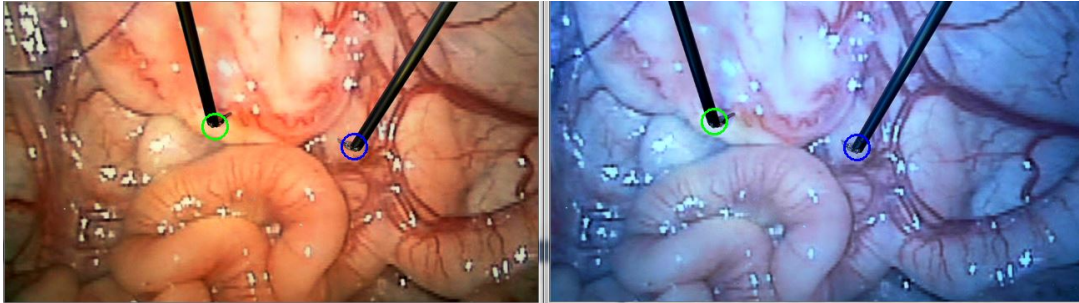


Figure 7.1: The detection of two surgical tools using the presented method.

Using the geometric constraint approach, a multi-threaded program was developed which could detect the tool tips in both channels simultaneously and then extract the depth and Cartesian coordinates using a disparity mapping. The disparity mapping model varies depending on the specific camera unit used. For this work an experimental webcam setup was initially used to fine tune the algorithm and evaluate the efficacy of such a method (Chapter 5). Eventually, a full endoscopic video unit from a da Vinci robot was utilized to test the performance of the algorithm in surgical conditions (Chapter 6). For both video sources used, the general method for tool tracking followed the same high-level methodology.

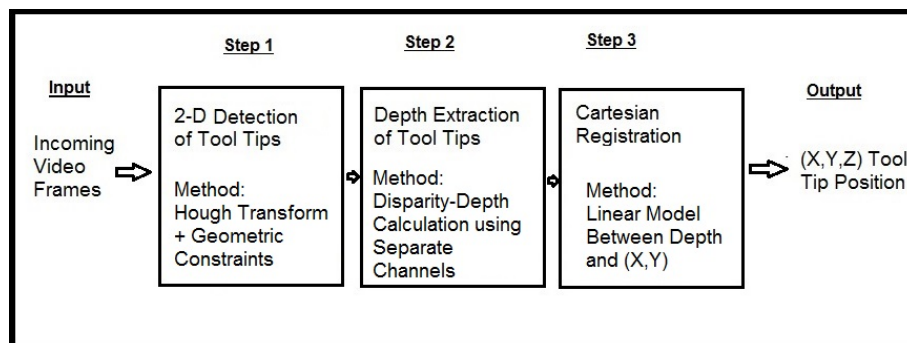


Figure 7.2: The general steps used for tracking surgical tools and the specific algorithms chosen for each step.

Using the experimental webcam setup, the model used for mapping disparity into depth values followed a relatively simple power function (Eq. 4.6). However, for the endoscopic camera setup, the depth mapping complexity was increased due to significant radial distortion. Instead of designing a complete ‘fisheye’ correction filter, a model was adopted in which the offset due to radial distortion was coupled into the disparity-depth mapping (Eq. 6.1, 6.2). Because of the extremely close interocular distance on the endoscope, a small range of disparities was available for a large range of depths. As a result there was a high signal to noise ratio when mapping disparities into depths.

For both the webcam and endoscope video sources, the model for extracting (X_w, Y_w) Cartesian coordinates was very similar. Once the depth has been calculated for the tool tips, the Cartesian coordinates are registered as a linear relationship with the product of the depth and the pixel location (Eq. 6.5, 4.7). For both the camera setups, the accuracy of this model was very good and contained little error even in the face of depth extraction errors.

7.2 Performance Analysis

From the outset, seven design goals were adopted for the develop of this algorithm (Fig. 3.1). Of these goals, the speed and accuracy of the tracking algorithm were decidedly the most important. A series of experiments and analyses were performed in order to determine the computational speed, the tracking noise, the tracking error, and the robustness of the tracking algorithm (Sec. 5.1). These experiments were performed on the tracking algorithm for both camera setups as well as a pre-recorded video from a live surgical procedure.

For the experimental webcam setup, the overall computational time was determined to be **39.9 ms** per frame. This computational time resulted in a working frame rate of **25.86 FPS**. The localization noise for a stationary tool tip was determined to be **1.29 mm**. In terms of accuracy, the tool tip was located within 8 mm of a known trajectory **99.4%** of the time. This correlates to a 95th percentile of tool reprojection errors within **5.47 mm**. The average reprojection error was found to be **3.05 mm**. The results from the experimental webcam setup were very promising and indicate a high level of efficacy in the algorithm used for tool tracking.

Table 7.1: The various performance metrics are compared for the webcam setup and the endoscope setup.

Performance Metric	Webcam Setup	Endoscope Setup
Computation Time (ms)	39.9	33.99
Frame Rate (FPS)	25.86	26.98
Depth Model Reprojection Error (mm)	4.09	7.89
Localization Noise (Total) (mm)	1.29	7.92
Average Overall Tracking Error (mm)	3.05	8.68
Average X-Y Tracking Error (mm)	1.59	1.88
95 th Percentile Error (mm)	5.47	15.06
Percent of Reprojections Within 8 mm	99.4 %	55.22 %

For the endoscopic camera setup, the results differed from the experimental webcam setup. In terms of computational speed each frame required **33.99 ms** to localize the tool tips. This computational time resulted in a frame rate of **26.99 FPS**. While this frame rate is slightly faster than the experimental webcam setup, it is likely related to slight fluctuations in PC performance as opposed to a change in the tracking algorithm. In terms of tracking noise for a stationary tool, the endoscopic camera setup had an average reprojection noise of **7.92 mm**. This can be attributed to the high signal to noise ratio in the depth extraction model. The tracking accuracy for the endoscopic camera setup was slightly worse than the experimental camera setup. The average reprojection error when compared with a known trajectory was found to be **8.68 mm**. Similarly, the percent of reprojections with 8 mm of the known trajectory was **55.22%**. The 95th percentile of all reprojections were within **15.06 mm**. While the accuracy performance using the endoscopic camera does not appear as strong as the result from the webcam setup, this data is still promising. While the average overall tracking error

was 8.68 mm, this error is largely due to the depth extraction model. In comparison, the average tracking errors in the X_w and Y_w components was only **0.35 mm** and **1.25 mm**, respectively. From this data we can infer that given a more accurate depth extraction model, that the performance for the endoscopic setup would be much improved.

Finally, in terms of robustness, the webcam setup was examined for tracking accuracy in the face of varied background textures and different lighting conditions. The tool tracking algorithm was still able to locate the tool tip. For the endoscopic camera, the tool tracking algorithm was applied to a pre-recorded video of a prostate removal procedure. From this video, a 54 second clip where the right and left tool were both in the field of view was used. Upon manual inspection of each frame in the clip, the tracking algorithm correctly located at least one tool **88.0%** of the time. The tool tracking algorithm also located both tool tips **51.2%** of the time. However, for **12.0%** of the frames, neither of the tool tips was correctly located. This result can be attributed in part to the presence of electrocautery smoke which partially occluded the tool tips during portions of the video clip (approximately 8% of the time). This result can also be attributed to the fact that for the particular surgical video used, the tool tips were kept extremely close to the endoscope lens which resulted in the portion of visible tool shaft being extremely short. Without a reasonable amount of tool shaft length visible, this tracking algorithm does not perform as well.

7.3 Performance Conclusion

The surgical tool tracking algorithm presented in this work provides very promising results in terms of accuracy and especially speed. For the experimental camera setup, the tool tracking algorithm sufficiently met all 7 design goals outlined in Chapter 3. The 99.4% successful tool localizations exceeds the design goal of 86%. In terms of speed, the working frame rate of 25.86 FPS exceeds the 16 Hz design goal. All other design goals are inherently met by the algorithm used. The tool tracking algorithm is agnostic to tool type, simple and easy to distribute, capable of providing accurate timing data and robust to changes in conditions. For the endoscopic camera setup, the working frame rate of 26.98 FPS still exceeds the prescribed design goal. However, the tracking algorithm does fall below our accuracy goal with only 55.2% tracking accuracy.

However, it is envisioned that future work on the depth extraction model may improve this performance to a sufficient level.

While the presented algorithm does meet many of the specified design goals, there are certain shortcomings which are openly acknowledged. The tool tracking algorithm cannot detect the tool tips accurately when they are either not in the field of view or are occluded. While certain probabilistic estimation procedures could have been added to mitigate this issue, that was not within the scope of this work. The tool tracking algorithm is also not capable of tracking or detecting the position or configuration of the tool wrist. While this is beneficial in that the algorithm is agnostic to wrist configuration, it does result in a decrease in useful information. However, this problem has already been partially resolved by White et al. who developed a mechanism for extracting wrist configuration information from surgical robotic tools [69]. Using this method they are able to compute the wrist configuration using a series of potentiometers and sensors on the wrist actuator. The presented work is also partially constrained in its requirement to have at least part of the tool shaft in the field of view. For procedures in which the field of view is focused only on the tool wrist, this shaft geometry-based tool tracking algorithm does not perform well. However, extending it to similarly structured geometric features of the wrist or end effectors may enable tracking in such situations while maintaining its favorable frame rates.

7.3.1 Future Work

In future iterations of the proposed method, it is intended that several of the issues will be resolved and re-designed. One of the most crucial aspects will be a re-design of the depth extraction method for the endoscopic camera setup. The performance characterization has revealed several shortcomings in terms of accuracy using the presented disparity-depth model. With the necessary documentation concerning the details of the da Vinci endoscope, it should be possible to develop a more comprehensive model for depth extraction using this camera setup.

Barring the availability of specifications for the da Vinci endoscope, future work may include a revised approach for extracting the depth and Cartesian coordinates of the surgical tool tip. One of the envisioned approaches for resolving this issue is to use the line separation of the tool to extract the depth. The diameter of a specific tool can

be easily measured and subsequently used to correlate the separation of the tool shaft in pixel space with Cartesian coordinates. Since closer tools will appear to have a wider shaft in pixel space, this information can be exploited to determine the depth to the tool tip.

Another area of future work is a standardization of the depth calibration routine. For this work the calibration was done manually at arbitrarily selected baseline depths. In order to make this algorithm more accessible to outside parties, a calibration program should be developed in which any camera source can be subjected to an automated calibration board detection scheme which can then compute the coefficients needed for the depth extraction model.

A third proposed addition in future iterations of this work will be an extension to laparoscopy and the tracking of laparoscopic tools. This is logical extension of this tracking algorithm since the tool shafts of laparoscopic tools have many visual similarities to surgical robotic tool shafts. Both are long, straight tool shafts whose colors are unique and distinct from the anatomical background. The necessary modifications to track laparoscopic tools with this algorithm would be very straightforward.

Finally, future iterations will likely involve fine tuning this algorithm to make it more robust to changing conditions. One such addition will likely include a probabilistic estimation routine in the case of lost or uncertain tool localizations. Using the current information provided by the tracking algorithm, there exists a logical extension in which lost or occluded tool position can be extrapolated based on prior information.

References

- [1] Richard Satava, Roger Smith, and Vipul Patel. Fundamentals of robotic surgery: Outcomes measures and curriculum development. In *Society of Laparoendoscopic Surgeons*, 2012.
- [2] Gary Null, Carolyn Dean, Martin Feldman, and Debora Rasio. Death by medicine. *Journal of Orthomolecular Medicine*, 20:21–24, 2005.
- [3] D. M. Herron and M. Marohn. A consensus document on robotic surgery. *Surgical Endoscopy*, 22:313–325, 2008.
- [4] Roger Smith, Vipul Patel, and Richard M. Satava. Simulation and surgical training: Fundamentals of robotic surgery. Powerpoint, August 2011.
- [5] Timothy N. Judkins, Dmitry Oleynikov, and Nick Stergiou. Objective evaluation of expert and novice performance during robotic surgical training tasks. *Surgical Endoscopy*, 23:590–597, 2009.
- [6] K Narazaki, D. Oleynikov, and N. Stergiou. Robotic surgery training and performance: Identifying objective variables for quantifying the extent of proficiency. *Surgical Endoscopy*, 20:96–103, 2006.
- [7] Malcolm G. Munro. Surgical simulation: Where have we come from? where are we now? where are we going? *The Journal Of Minimally Invasive Gynecology*, 19:272–283, 2012.
- [8] S. A. Fraser, D. R. Klassen, L. S. Feldman, G. A. Ghitulescu, D. Stanbridge, and G. M. Fried. Evaluating laparoscopic skills. *Surgical Endoscopy*, 17:964–967, 2003.

- [9] M.P. Schijven, J. Jakimowicz, and C. Schot. The advanced dundee endoscopic psychomotor tester (adept) objectifying subjective psychomotor test performance. *Surgical Endoscopy*, 16:943–948, 2002.
- [10] A.G. Gallagher, A.B. Lederman, K. McGlade, R. M. Satava, and C. D. Smith. Discriminative validity of the minimally invasive surgical trainer in virtual reality (mist-vr) using criteria based on expert performance. *Surgical Endoscopy*, 18:660–665, 2004.
- [11] Roger Kneebone. Simulation in surgical training: educational issues and practical implications. *Medical Education*, 37(3):267–277, FEB 2003.
- [12] Nicholas Stylopoulos, Stephane Cotin, Steven Dawson, Mark Ottensmeyer, Paul Neumann, Ryan Bardsley, Micheal Russell, Patrick Jackson, and David Rattner. Celts: A clinically-based computer enhanced laproscopic training system. In James D. Westwood, editor, *Medicine Meets Virtual Reality 11: NextMed : Health Horizon*, 2003.
- [13] Justin M. Albani and David I. Lee. Virtual reality-assisted robotic surgery simulation. *Journal of Endourology*, 21:285–287, 2007.
- [14] Michelle A. Lerner, Mikias Ayalew, William Peine, and Chandru Sundaram. Does training on a virtual reality robotic simulator improve performance on the da vinci surgical system? *Journal of Endourology*, 24:467–472, 2010.
- [15] Intuitive Surgical. www.intuitivesurgical.com, October 2012.
- [16] L. W. Sun, F. Van Meer, J. Schmid, Y. Bailly, A. A. Thakre, and C. K. Yeung. Advanced da vinci surgical system simulator for surgeon training and operation planning. *International Journal Of Medical Robotics and Computer Assisted Surgery*, 3:245–251, 2007.
- [17] Ankur Baheti, Sridhar Seshadri, Amrish Kumar, Govindarajan Srimathveeravalli, Thenkurussi Kesavadas, and Khurshid Guru. Ross: Virtual reality robotic surgical simulator for the da vinci surgical system. In *Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems*. IEEE, 2008.

- [18] Mark W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. John Wiley & Sons Inc, 1989.
- [19] Jeffrey H.Peters, Gerald M.Fried, Lee L.Swanstrom, Nathaniel J. Soper, Lelan F. Sillin, Bruce Schirmer, and Kaaren Hoffman. Development and validation of a comprehensive program of education and assessment of the basic fundamentals of laparoscopic surgery. *Surgery*, 135:21–27, 2004.
- [20] Thomas S. Lendvay, Pasquale Casale, and Robert Sweet. Initial validation of a virtual-reality robotic simulator. *Journal of Robotic Surgery*, 2:145–149, 2008.
- [21] Krishna Moorthy, Yaron Munz, Supid Sarker, and Ara Darzi. Objective assessment of technical skills in surgery. *British Medical Journal*, 327:1032–1037, 2003.
- [22] Tim Kowalewski, Lee White, Thomas S. Lendvay, Iris Jiang, Andrew Wright, and Blake Hannaford. Validation of edge, a reality-based laparoscopic box trainer for quantitative psychomotor skill assesment. *Journal of the Society for Simulation in Healthcare*, 2012.
- [23] David M. Kwartowitz, S. Duke Herrell, and Robert L. Galloway. Toward image-guided robotic surgery: Determining intrinsic accuracy of the da vinci robot. *International Journal of Computer Assisted Radiology and Surgery*, 1:157–165, 2006.
- [24] Carol E. Reiley, Henry C. Lin, David D. Yuh, and Gregory D. Hager. Review of methods for objective surgical skill evaluation. *Surgical Endoscopy*, 25:356–366, 2011.
- [25] Tracey A. Morley. Roll-pitch-roll surgical tool, 2004.
- [26] M.K. Chmarra, C.A. Grimbergen, and J. Dankelman. Systems for tracking minimally invasive surgical instruments. *Minimally Invasive Therapy*, 16:6:328–340, 2007.
- [27] Walter T. Appleberry, 1982.
- [28] G. B. Hanna, T. Drew, P. Clinch, B. Hunter, and A. Cuschieri. Computer-controlled endoscopic performance assesment system. *Surgical Endoscopy*, 12:997–1000, 1998.

- [29] A. S. Morris, M. Dickinson, and A. M. S.Zalzala. An enhanced ultrasonic system for robot end effector tracking. In *International Conference on Control*, 1991.
- [30] Florin Tatar, Jeff Mollinger, Jeroen Bastemeijer, and Andre Bossche. Micro precision achieved for measuring the position of surgical tools using two-frequency method. In *IEEE Sensors*, 2005.
- [31] C. Sokollik, J. Gross, and G. Buess. New model for skills assesment and training progress in minimally invasive surgery. *Surgical Endoscopy*, 18:495–500, 2004.
- [32] Zebris GmbH. www.zebris.de, October 2012.
- [33] Rasool Khadem, Clement C. Yeh, Mohammad Sadeghi-Tehrani, Michael R. Bax, Jeremy A. Johnson, Jacqueline Nerney Welch, Eric P. Wilkinson, and Ramin Shahidi. Comparitive tracking error analysis of five different optical tracking systems. *Computer Aided Surgery*, 5:98–107, 2000.
- [34] Christoph A. Amstuts, Nikolaos E. Bechrakis, Micheal H. Foerster, Jens Heufelder, and Jens H. Kowal. Intraoperative localization of tantalaum markers for proton beam radiation of choroidal melanoma by an opto-electronic navigation system: A novel technique. *International Journal of Radiation Oncology, Biology, Physics*, 82:1361–1366, 2012.
- [35] Guo-Qing Wei, Klaus Arbter, and Gerd Hirzinger. Automatic tracking of laparoscopic instruments by color coding. In *Medical Robotics and Computer-Assisted Surgery*, pages 357–366, 1997.
- [36] Alexandre Krupa, Jacques Gangloff, Christophe Doignon, Michel de Mathelin, Guillaume Morel, Joel Leroy, Luc Soler, and Jacques Marescauz. Autonomous 3-d positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. *IEEE Transactions on Robotics and Automation*, 19:842–853, 2003.
- [37] Martin Groeger, Klaus Arbter, and Gerd Hirzinger. *Motion Tracking for Minimally Invasive Robotic Surgery*. I-Tech Education and Publishing, 2008.
- [38] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems Man and Cybernetics*, 9:62–66, 1979.

- [39] Cheolwhan Lee and Derrin Uecker. Image analysis for automated tracking in robot-assisted endoscopic surgery. *Pattern Recognition*, 8:88–92, 1994.
- [40] Christopher Doignon, FlorentNagcotte, and Michel De Mathelin. Detection of grey regions in color images: Application to the segmentation of a surgical instrumenty in robotized laparoscopy. In *Proceedings of the IEEE/RSJ Conference on INtelligent Robotics and Systems*, 2004.
- [41] Christopher Doignon, Pierre Graebing, and Michel De Mathelin. Real-time segmentation of surgical instruments inside the abdomonal cavity using a joint hue saturation color feature. *Real-Time Imaging*, 11:429–442, 2005.
- [42] Christopher Doignon, Florent Nageotte, and Michel De Mathelin. Segmentation and guidance of multiple rigid objects for intra-operative endoscopic vision. *Dynamical Vision*, pages 314–327, 2007.
- [43] Sandrine Voros, Jean-Alexandre Long, and Philippe CinQuin. Automatic detection of laparoscopic images: A first sep ttoward high-level command of robotic endoscopic holders. *The International Journal of Robotics Research*, 26:1173–1190, 2007.
- [44] Paul V.C. Hough. Method and means for recognizing complex patterns, 1962.
- [45] Remi Wolf, Josselin Duchateau, Philippe Cinquin, and Sandrine Voros. 3d tracking of laparoscopic instruments using statistical and geometric modeling. In *Medical Image Computing and Computer Assisted Intervention*, 2011.
- [46] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference On Computer Vision and Pattern Recognition*, 2001.
- [47] Austin Reiter, Peter K. Allen, and Tao Zhao. Feature classification for tracking articulated surgical tools. In *Medical Image Computing and Computer Assisted Intervention*, 2012.

- [48] Zachary Pezzementi, Sandrine Voros, and Gregory D. Hager. Articulated object tracking by rendering consistent appearance parts. In *IEEE International Conference on Robotics and Automation*, 2009.
- [49] Open Graphics Library. <http://www.opengl.org/>.
- [50] Marco Feuerstein, Tobias Reichl, Jakob Vogel, Joerg Traub, and Nassir Navab. Magneto-optical tracking of flexible laparoscopic ultrasound: Model-based online detection and correction of magnetic tracking errors. *IEEE Transactions on Medical Imaging*, 28:951–967, 2009.
- [51] Northern Digital. <http://www.ndigital.com/medical/aurora-faqs.php>, October 2012.
- [52] Willow Garage. <http://opencv.willowgarage.com/wiki/>, October 2012.
- [53] Anshul Thakral, Jeffrey Wallace, Damian Tomlin, Nikesh Seth, and Nitish Thakor. Surgical motion adaptive robotic technology (s.m.a.r.t): Taking the motion out of physiological motion. *Medical Image Computing and Computer-Assisted Intervention*, 2208:317–325, 2001.
- [54] MTI Instruments. <http://www.mtiinstruments.com/products/fiberopticmeasurement.aspx>, October 2012.
- [55] Yoshihiko Nakamura, Kosuke Kishi, and Hiro Kawakami. Hearbeat synchronization for robotic cardiac surgery. In *International Conference on Robotics & Automation*, 2001.
- [56] Tobias Ortmaier, Martin Groger, Dieter H. Boehm, Volkmar Falk, and Gerd Hirzinger. Motion estimation in beating heart surgery. *IEEE Transactions on Biomedical Engineering*, 52:1729–1740, 2005.
- [57] R. Ginhoux, J. A. Gangloff, M.F. de Mathelin, L. Soler, Mara M. Arenas Sanchez, and J. Marescaux. Beating heart tracking in robotic surgery using 500 hz visual servoing, model predictive control and an adaptive observer. In *International Conference on Robotics & Automation*, 2004.

- [58] Shelten G. Yuen, Paul M. Novotny, and Robert D. Howe. Quasiperiodic predictive filtering for robot-assisted beating heart surgery. In *IEEE International Conference on Robotics and Automation*, 2008.
- [59] Holger Monnich, Heinz Worn, and Daniel Stein. Op sense - a robotic research platform for telemanipulated and automatic computer assisted surgery. In *IEEE International Workshop on Advanced Motion Control*, 2012.
- [60] Mark Young, Erik Beeson, James Davis, Szymon Rusinkiewicz, and Ravi Ramamoorthi. Viewpoint-coded structured light. UCSC, 2007.
- [61] Ethan D. Grober, Stanley J. Hamstra, Kyle R. Wanzel, Richard K. Reznick, Edward D. Matsumoto, Ravindar S. Sidhu, and Keith A. Jarvi. The educational impact of bench model fidelity on the acquisition of technical skill: The use of clinically relevant outcome measures. *Annals of Surgery*, 240:374–381, 2004.
- [62] Christian R. Larsen, Jette L. Soerensen, Teodor P. Grantcharov, Torur Dalsgaard, Lars Schouenborg, Christian Ottosen, Torben V. Schroeder, Bent S. Ottesen, and Juliane Marie Centre. Effect of virtual reality training on laparoscopic surgery randomised controlled trial. *British Medical Journal*, 338:1–6, 2009.
- [63] Jacob Rosen, Massimiliano Solazzo, Blake Hannaford, and Mika Sinanan. Objective laparoscopic skills assessments of surgical residents using hidden markov models based on haptic information and tool-tissue interactions. *Studies in Health Technology and Informatics - Medicine Meets Virtual Reality*, 81:417–423, 2001.
- [64] Qt project. <http://qt-project.org/>, 2013.
- [65] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 2000.
- [66] J Matas, C Galambos, and J Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer Vision and Image Understanding*, 78:119–137, 2000.
- [67] Paul Viola and Michael Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

- [68] Roger Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, 3:323–344, 1987.
- [69] Lee Woodruff White, Timothy Mariusz Kowalewski, Thomas S. Lendvay, and Blake Hannaford. Motion and video capture for tracking and evaluating robotic surgery and associated systems and methods, 2012.
- [70] Arduino. www.arduino.cc.

Appendix A

Software Details

A.1 Software Implementation Details

The use of a global variables file was implemented in order to share common knowledge variables throughout the program threads. The ‘globals.cpp’ file contains several common variables including boolean values for when to start and stop the application so that every thread can begin and exit safely. The global information also includes frame size and other camera information such as frame rate. In addition to this a global variable is allotted in order to share data about the last known location of the tool tip in pixel space so that any thread may query that information. The final key global variable is a set of ‘Mutex’ handles. ‘Mutex’ handles are used to keep memory safe when it is being accessed by multiple threads simultaneously. Mutexes work by ‘locking’ a particular location in the computers memory while a particular thread accesses the information stored there. While the accessing thread or ‘owning’ thread has control over that particular location, other threads cannot access that information. Once the owning thread has released or ‘unlocked’ that memory location, then the next thread may ‘lock’ and access that data. Mutex handles don’t operate automatically and need to be explicitly written into a program whenever a bit a code attempts to access a shared variable. In the case of this work the Mutex handles are used to protect data that gets sent from one thread to subsequent threads in the form of an array. This code uses the Windows API Mutex implementation which initializes Mutexes using the ‘CreateMutex()’ function call.

Another important note is the application uses the C++ standard ‘time.h’ functionality in order to get time stamps throughout the application. The ‘time.h’ class provides the ‘clock()’ function which simply returns a millisecond value since the program started. This value is stored in a variable of type ‘clock_t’. The ‘clock_t’ variables can then be cast as double precision in order to save it to a data file. Such time stamps are important for calculating computational performance and benchmarking framerates. In the supporting threads a begin time variable is called in the constructor, then when a timestamp is required the difference between the current time and the start time is taken. The reason for the relative time stamp is that not all threads are created and start at the same time so a small relative offset is necessary.

A.1.1 Thread Creation

In order to setup secondary threads in the application, the main thread first defines function handles as ‘winapi’ functions. These function handles are then given as input arguments to the standard ‘CreateThread()’ function. This opens up the secondary thread functions to run simultaneously and also returns a handle to that thread in the form of a temporary variable. This handle is saved in memory until the program exits and the ‘CloseHandle()’ function is called in order to completely close down the supporting threads. The 5 threads created are for the following functions; video capture, tool tracking right, tool tracking left, 3D localization, and the supporting GUI (Fig. 4.25). Each of the 5 thread functions are wrapped in Try/Catch blocks so that if the particular computer in use cannot handle complex multi-threading, the application will return an error and close down.

The supporting threads also require the use of the signal events that can be used to ensure proper exiting and closing of the application. The signal events are initialized in the main function using the ‘CreateEvent()’ function. These events are then signaled by the supporting threads upon exit using the ‘SetEvent()’ function. The ‘main()’ then utilizes the ‘WaitForMultipleObjects()’ function in order to prevent the application from closing until all threads have safely exited. After all events have been signaled the thread handles can be closed.

A.1.2 Custom Data Structures

The first struct type labeled ‘MultiFrame’ contains 3 data types; an OpenCV Mat image, a time stamp of type ‘double’, and frame index of type ‘int’. The ‘MultiFrame’ structure is used to share the captured stereo video frames between the capture thread and the two tracking threads. The second struct type is entitled ‘LocateFrame’ and contains 11 variables. The first variables are the x_p coordinates for the each of the tool tips, of type ‘int’. The second variables are the y_p coordinates, also of type ‘int’. Data variables 3-5 in the structure are of type ‘double’ and correspond to the clock time when a frame was captured, when the tracking thread started work on that frame, and when it finished. The other variables include an ‘int’ frame index corresponding to the capture frame, an ‘int’ representing the number of tools detected in that frame, and a boolean indicating which if any tools were lost. The struct also includes 3 ‘double’ type variables corresponding to the length, the distance to prior locations, and the theta values of each tool found in the frame.

In total, 4 ‘deque’ buffers are initialized; 2 ‘MultiFrame’ buffers (one for each stereo video channel) and 2 ‘LocateFrame’ (one for each tracking thread to send it’s pixel location information). These containers are then passed by reference to the 4 corresponding threads where a copy is made so that a particular thread can either stuff information into the buffer or extract data out. Each time one of the threads either appends or extracts data from the buffer, that thread must first wait to lock the mutex related to that particular ‘deque’ buffer. There are 4 mutexes which exist for this purpose: ‘Mutexframeright’, ‘mutexframeleft’, ‘mutexlocateright’ and ‘mutexlocateleft’. The final purpose of the ‘main()’ function before closing the application is to clear the 4 ‘deque’ buffers and delete the memory locations for those buffers.

The capture thread takes as input arguments the two ‘MultiFrame’ buffers: ‘frameQueueR’ and ‘frameQueueL’. These buffers are copied to local pointers in the ‘CaptureClass’ constructor. The same buffer pointers are deallocated in the classes destructor. After the capture class has been initialized, the thread is used to perform the run function. The ‘runCapture()’ function first attempts to open up two video capture sources attached to the computer. The video capture sources are initialized using the OpenCV ‘VideoCapture’ object and are specified as the right capture and the left capture. Each video capture object then calls the ‘open()’ method to open up a video capture stream.

In OpenCV the video sources are enumerated from 0-9. Source 0 refers to the systems default source and sources 1 and higher are then arbitrarily assigned. As a result of the undefined nature of the OpenCV capture sources, the enumerations are currently set to 0 and 1 respectively for the right and left cameras in the stereo setup. However, if there are additional cameras attached to the computer, the user must change a configuration file in order to change the camera enumerations. Once the default camera enumerations have been opened the code checks to make sure the enumerations have actually succeeded in opening two unique sources. If the capture sources failed to initialize the application will return an error and close.

The capture thread will run until another thread signals an exit to the application or a capture device is unplugged from the computer, resulting in an error. Once the while loop has exited the right and left capture objects are deallocated and the function exits.

The right tracking thread takes as input a pointer to both a ‘MultiFrame’ and a ‘LocateFrame’ buffer line those described in section 4.2.1. The ‘MultiFrame’ buffer is the same pointer as the ‘frameQueueR’ buffer given to the capture thread while the ‘LocateFrame’ buffer is labeled ‘locateQueueR’ and is also passed to the 3D localization thread. Similarly the left tracking thread takes as inputs the pointer to the ‘frameQueueL’ buffer as well as the ‘locateQueueL’ buffer. in both the left and right tracking constructors, a copy is made of these buffer pointers.

A.1.3 Tracking Thread Functions

Both tracking classes contain predefined functions for distance formula, line separation and cross products. These functions follow the equations discussed earlier in section 4.1 and therefore do not deserve a redundant explanation. Each tracking thread also contains a build in function for recording video clips. These functions use the OpenCV ‘VideoWriter’ object to save individual frames to an ‘.avi’ video file for offline analysis of the tracking algorithm.

Hough line data along with pixel location information is stored in both tracking threads in custom data structure entitled ‘ToolTips’. This structure is then passed to various functions in the tracking class in a vector of ‘ToolTips’ structures in order to avoid confusing and prolonged input arguments. The ‘ToolTips’ structure contains

information such as pixel location, previous locations, distance to prior location, line theta angles, line lengths, and whether the tool tip has been lost or found.

As in the capture class the primary functionality of the tracking threads is found in the ‘runTrack()’ function. In the run function OpenCV windows are created so that once the tool tip has been tracked, a circle can be superimposed on the frame and displayed for the user. Next the vectors of ‘ToolTips’ structures is initialized so that each data member is set to some initial value; usually zero. Next all other variables are initialized and OpenCV Mat containers are declared. It is also important to note that the previous location for the tool tip is initialized to center of the image (in the case of 1 tool being tracked) or the center of the right half of the image and the center of the left half of the image (in the case of 2 tools being tracked). The time stamp variables are also initialized so that the time at the start and end of each object detection loop can be recorded. Once all initialization has taken place, the run function enters a continuous while loop.

In order to extract the oldest ‘MultiFrame’ structure from the buffer, both tracking threads first use the ‘WaitForSingleObject()’ function to lock their respective ‘deque’ buffer. Once a lock has been acquired, a ‘MultiFrame’ structure is copied to a local variable. The oldest object in the buffer is then deleted and the buffer shifted. The index and time stamp corresponding to this frame are stored so that they can be transferred to the 3D localization function.

A.1.4 3D Localization Thread Functions

The first step in the 3D localization thread is to acquire the information from the right and left tracking threads. These threads having simultaneously detected the tool tip pixel position have appended this information in the form of ‘LocateFrame’ structures to a shared buffer. Each tracking thread uses a separate buffer, both of which are then accessed by the 3D localization thread. In order to acquire the buffer object for both the right and left tracking data, this thread must acquire two separate mutexes at once; one for the right ‘LocateFrame’ buffer and one for the left ‘LocateFrame’ buffer. Once both mutexes have been acquired using the ‘WaitForSingleObject()’ function, the data from each buffer can be extracted. The oldest ‘LocateFrame’ structure in each buffer is assigned to a local variable of the same structure type and then the oldest element

in the buffer is deleted from each channel. The structure data is then first analyzed in terms of frame index. The occurrence asynchronous frame analysis is very infrequent since the computational time should be the same for both channels, however the frame index check is used as a fail safe.

After acquiring right and left location information with the same index, several time stamps from throughout the tracking process are saved so they can be used for benchmarking. These include; the time stamp when the frame was captured for each channel, the time stamp when each channel started the object detection algorithm, the time stamp once each channels tracking detection had determined the pixel location, and the time stamp once the 3D localization algorithm had started.

The implementation of the (X_w, Y_w, Z_w) equations is very straightforward and only requires the use the standard C++ 'power' function which allows values to be raised to a certain power.

Appendix B

Open Medical Interface

B.1 Initial Design Specification

The Open Medical Interface (OMI) has been designed as standardized protocol for interaction between a wide assortment of hardware devices in the medical field. This protocol allows devices with no prior knowledge of the other to transfer standard data strings back and forth with minimal setup requirements. The envisioned uses for this design include medical training devices, imaging systems, medical devices and other hardware based system ubiquitous in the medical field. This interface then allows these devices to communicate with software based systems such as online viewing and training systems, PC user interfaces, database systems and hospital data management systems. Such an interface systems even allows for inter-hardware communication like that between an imaging system and the medical training device.

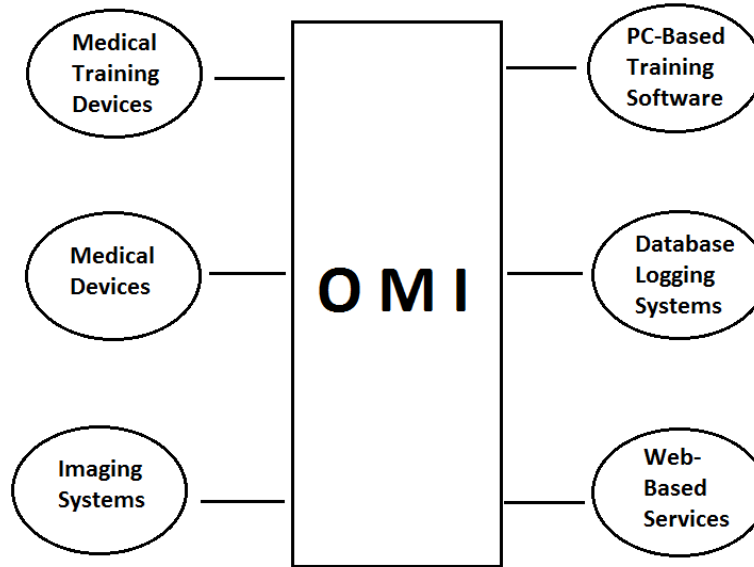


Figure B.1: Overview of the OMI design.

In its initial implementation the OMI protocol has been designed as library written in the C++ programming language for easy inclusion into micro-processor code such as that found in the Arduino development environment. While this is the specific environment used for design and testing, the basic system components can and will be transitioned into a multitude of environments such as Java/mobile devices, web-based devices, Serial Peripheral Interface (SPI) Bus systems, and Bluetooth devices. The priority of the design is not the specific language or implementation, but the basics of the data transfer.

The main data packet format used by the system is a comma delimited data packet with both a custom header character and a standard end packet character (Fig. B.2). For the purposes of testing this header character was 'H' and the end character was 'Q'. Between each comma separation, any alpha-numerical string combination can be included. The second member of the data packet is used as a custom letter for identifying the subsequent data members. This identifier is known by both the sender and the receiver. The quantity of individual strings is not important so long as the end and

start characters are the standard identifiers. On the receiving end, the listening port will query the data stream until a packet meeting the header specification is received. Once received the port will parse through the data packet and store the individual strings until it reaches the end character. However if the end-character is not found in the packet, the stored data will be discarded, thus ensuring complete data transfer.

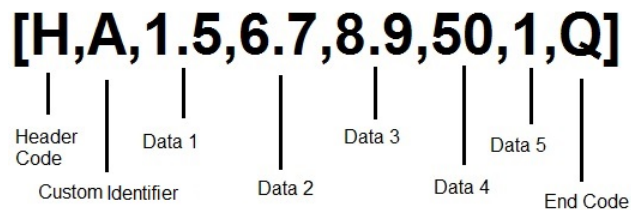


Figure B.2: Structure of the OMI data packet.

The primary idea behind the library is that this data transfer can be used to transfer names, titles, identification codes as well as numerical data. In an initial test of the library usage, a graphical user interface (GUI) was written in C++ to sit on the computer and receive commands from the library which was housed on an Arduino board. The board and the computer were connected via USB serial. The GUI would initially display a blank screen, but would query commands from the serial port. The library would then be used to send titles, names and data values which would then be updated on the screen. In this fashion, a hardware device could be used in conjunction with a GUI without the user ever programming the GUI itself.

The Open Medical Interface was devised as a means for standardizing data flow within medical devices and surgical training devices. While not directly applicable to the surgical tool tracking method presented in this work, this standard will be beneficial in the design of future medical training devices.

Appendix C

SurgTRAK Development

C.1 Overview

SurgTRAK is an open source environment designed to provide easy data acquisition for the development and analysis of medical devices and equipment. The SurgTRAK environment allows easy interaction with electro-mechanical sensors, video acquisition systems and other data recording devices. These devices are then displayed in an all encompassing user interface which allows for synchronized data collection between all sensors.

The SurgTRAK environment was initially designed at the University of Washington BioRobotics lab. The software is written in C++ with several external libraries used for data acquisition. The data sources can come from Phidgets boards (Phidgets Inc, Calgary, Alberta), Arduino boards [70] and video sources to name a few. As a secondary portion of the surgical tool tracking works, the SurgTRAK program was modified to utilize the OpenCV library. The purpose of this was to incorporate the tool tracking algorithm into the SurgTRAK environment. The aim of this addition was to allow surgeons or researchers to upload surgical videos from a PC running the SurgTRAK software or using a web page interface and receive feedback on performance metrics.

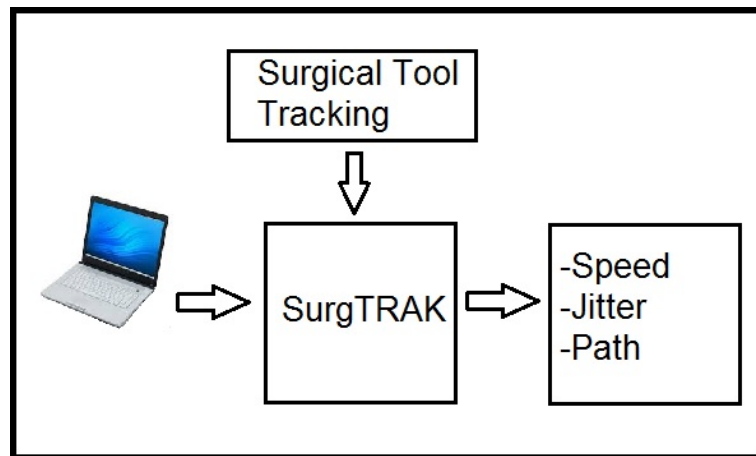


Figure C.1: Surgical video is uploaded to SurgTRAK from a computer and the performance is evaluated using the surgical tool tracking method.

The SurgTRAK environment is provided as an example of a method for distributing the surgical tool tracking algorithm. Ideally the source code for this algorithm will be made available for other researchers or developers to incorporate surgical tool tracking into their website or software environment. The SurgTRAK repository is currently still being updated and the incorporation of the surgical tool tracking algorithm with SurgTRAK is a work in progress.

Appendix D

Glossary and Acronyms

Care has been taken in this thesis to minimize the use of jargon and acronyms, but this cannot always be achieved. This appendix defines jargon terms in a glossary, and contains a table of acronyms and their meaning.

D.1 Glossary

- **Computer Vision (CV)** – A subsection of the computer science discipline which incorporates images and video with computer algorithms for performing tracking and other automated procedures.
- **da Vinci (DV)** – A surgical robot produced by Intuitive Surgical (Sunnyvale, CA) and is currently the only surgical robot with federal approval in the United States.
- **Hough Transform** – A computer vision algorithm which detects lines or other geometric patterns in images.
- **Edge Detection** – A computer vision algorithm which finds areas of high pixel variation and categorizes those areas as edges.
- **Multi-Threading** – A technique in computer programming which also multiple processes or commands to be processed simultaneously by the computer processor.

- **Mutex** – A variable used in computer programming to lock a certain location in a computer's memory so that only a single thread can access that information at a time.
- **Stereo** – Stereo video refers to a camera setup in which two cameras are mounted next to each other at a fixed distance away with parallel optical axes.
- **Optical Axis** – The imaginary line emanating from the center of the camera lens down the field of view.
- **End-Effector** – The tip or wrist of a robotic arm. In the case of a surgical robot this is usually a small metal grasper.
- **Mat** – The OpenCV container for images. This container allows access to individual pixels and other image manipulation.
- **Edge Detection** – The process of identifying strong edges in images by analyzing variations in pixel intensity.
- **Sobel (Edge Detection)** – An edge detection method which uses a kernel convolution which approximates a first order derivative.
- **Canny (Edge Detection)** – An edge detection method which utilizes a multi-directional derivative approach followed by hysteresis thresholding.
- **Probabilistic Hough Transform** – A faster implementation of the Hough transform line detection algorithm which uses only a subset of a typical edge sample.
- **First-In, First-Out** – A type of buffer or queue in which new data is added to the buffer at one end and removed from the other so that the oldest information is removed first.
- **Deque** – A dynamically created array which allows data to be appended onto the end and removed from the front or vice versa. This is a standard member of the C++ language.
- **Structure** – A structure or 'struct' is a custom defined container for holding multiple types of information. A struct allows code to pass a single variable which

can contains many different types of information such as numbers, letters, and/or images.

- **Double** – A C++ variable which is short for double precision floating point. A double contains a numerical value with a precision of 64 bits or rather a number with 16 digits past the decimal point.
- **Int** – A C++ variable which is short for integer. An 'int' contains no digits past the decimal points but can be an integer between 2,147,483,647.

D.2 Acronyms

Table D.1: Acronyms

Acronym	Meaning
MIS	Minimally Invasive Surgery
RMIS	Robotic Minimally Invasive Surgery
FRS	Fundamentals of Robotic Surgery
CV	Computer Vision
FPS	Frames Per Second
OR	Operating Room
OMI	Open Medical Interface
PHT	Probabilistic Hough Transform
CSV	Comma Separated Variable
GUI	Graphical User Interface
FIFO	First-in, First-out