ABSTRACT

GIALLO, JOSEPH FRANCIS, II. A Medical Robotic System for Laser Phonomicrosurgery. (Under the direction of Edward Grant.)

Phonomicrosurgery is a suite of complex otolaryngological surgical techniques related to the vocal folds. The primary operative challenges involve size and scaleability particularly when the carbon dioxide surgical laser is the operative tool of choice. The prevalent traditional methodology for remote control of the surgical laser is the mechanical micromanipulator. This device is capable of accurate laser aiming but is prone to error resulting from inexperience and ergonomic factors. Extensive training is required to employ the manual micromanipulator effectively.

Many of the difficulties associated with use of the mechanical micromanipulator are rooted in the ergonomics of the device. By necessity it is located in a disadvantageous position, i.e., attached to the base of the surgical microscope. As a result, the clinician has no convenient way of steadying his/her hand while making the precise, delicate movements necessary to accurately and consistently aim the surgical laser. The fact that the required movements are relatively small in nature exacerbates the accuracy and consistency problem.

The purpose of this dissertation is to document the design, development and application of a medical robotic system intended to improve this man machine interface. The primary goal of this research is the development of a device that moderates the operational challenges inherent in the classic manual micromanipulator, thereby enabling advances in clinical accuracy and efficiency.

A Medical Robotic System for Laser Phonomicrosurgery

by
Joseph Francis Giallo, II

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Biomedical Engineering

Raleigh, NC

2008

APPROVED BY:

Edward Grant, PhD Chair of Advisory Committee	H. Troy Nagle, PhD, MD
Charles Finley, PhD	Robert Buckmire, MD

David Lalush, PhD

DEDICATION

This work is dedicated to Joseph and William.

You can become anything you wish to be.

BIOGRAPHY

Joseph F. Giallo, II was born in Endicott, NY and is the eldest of three children. His family moved to Raleigh, NC when he was four years of age. Growing up in the local area he went on to undergraduate studies in Electrical Engineering at North Carolina State University. After earning his BSEE degree Joe worked in the telecommunications industry for seventeen years. He progressed through a series of increasingly senior roles in product development, product marketing, product line management and sales. During that time he also earned an MBA from the executive program at the Kenan Flagler School of Business at the University of North Carolina at Chapel Hill. He has worked in a large multi-national organization as well as two startup companies.

When not busy with biomedical engineering pursuits Joe enjoys full scale aviation and sharing his love of flying with others through flight instruction. He resides in Raleigh, NC with his wife of twenty years and their two sons, along with the family cocker spaniel.

ACKNOWLEDGEMENTS

First I would like to thank my parents for their love and support. As my first teachers they imparted the most important and timeless lessons. Chief among these is the fact that hard work and perseverance ultimately overcome all obstacles.

To my father, whose guidance over many years has taught me the value of patience, attention to detail, and the mindset that "if something is worth doing, it is worth doing right". I am also deeply indebted to him for his contributions and teachings in the area of precision machine work and fixture design and development. Without his assistance this project would not have been possible.

To my advisor, Dr. Edward Grant, who frequently provided excellent ideas, support, and counsel for the many challenges that I encountered during this project.

To Dr. Robert Buckmire, who was always generous with his time in educating me about phonomicrosurgery as well as enabling the numerous experiments that were vital to the success of this project. His enthusiasm for the project and its potential were always motivating to me, particularly when things were not going well.

To Dr. Charles Finley, an excellent engineer and always available "sounding board". I appreciate his willingness to invest substantial amounts of his personal time and energy helping me to work through various challenges and problems, as well as hosting me in his lab.

Last, and most important, to my wife Mary who has always supported me in whatever I have chosen to do. She made this achievement possible.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. A Review of Current User Interfaces for Laser Phonomicrosurgery

CHAPTER 2. Design of a Medical Robotic System for Laser Phonomicrosurgery

CHAPTER 4. Towards the Automation of Laser Phonomicrosurgery using a Medical Robotic System

Appendix IX. Mechanical fabrication process

LIST OF TABLES

CHAPTER 1. A review of current user interfaces for laser phonomicrosurgery

CHAPTER 2. Design of a Medical Robotic System for Laser Phonomicrosurgery

CHAPTER 3. The Testing and Evaluation of a Medical Robotic System for Laser Phonomicrosurgery

CHAPTER 1. A Review of Current User Interfaces for Laser Phonomicrosurgery

Abstract— Laser phonomicrosurgery is a demanding surgical technique requiring significant psychomotor skills. Scaleability, operative distance, and the anatomically small nature of the vocal folds all combine to create numerous surgical challenges. Currently the dominant user interface for remotely aiming a CO_2 surgical laser is the manual micromanipulator. This device is capable of accurate laser aiming but is prone to error resulting from inexperience and ergonomic factors. This paper explores the current state of the art with regard to user interfaces for laser phonomicrosurgery and proposes a novel research avenue to enhance this user interface.

I. INTRODUCTION

The first visible light laser was created by Theodore Maiman in 1960[1] and was based upon a ruby crystal. Over the next decade variations upon this fundamental technology were developed for physics and engineering applications [2] until ultimately the CO_2 laser was created for surgical research in 1970[3]. Surgical lasers are employed to cut or ablate tissue with a varying degree of coagulation. As a result, lasers based on a variety of different technologies have been developed for use in medicine, where the individual laser technology depends upon its intended use, e.g., the Nd:YAG (Neodymium:Yttrium Aluminum Garnet) laser is employed in tissue cutting and ablative applications while an Argon laser is more suited for tissue coagulation. Although lasers are used in many areas of medicine today, their use began with dermatology in 1962. In that year a dermatologist by the name of Dr. Leon Goldman first used laser technology to remove an unwanted tattoo. Dr. Goldman's accomplishment is widely recognized as the first medical application of laser technology. Lasers in medicine have since continued to evolve both in terms of underlying technologies and also in applications. In addition to dermatology[4] lasers have found widespread applications in ophthalmology, gynecology, and otolaryngology. The following section gives a brief overview of some of the medical application of lasers, organized by medical specialization.

Ophthalmologists use lasers in a variety of treatment protocols. The typical system employs a microscope coupled to a laser source and relies upon the highly developed psychomotor skills of the clinician to accomplish an acceptable operative result. For example, in the case of treating diabetic retinopathy there may be several thousand lesions per eye that must be created using the laser. Currently the clinician is responsible for manually determining the size and placement of each lesion as well as aiming and firing the laser to create each lesion. This is a tedious and time consuming process, one that is also error prone. Automation of this procedure by means of an advanced user interface giving fine position control will afford significant clinical benefit in terms of patient outcome as well as decreased operative time and expense. Other examples of conditions that are treated include intraocular pressure abnormalities due to glaucoma, and corneal shaping to improve vision [5]. One of the major opportunities for advancement in laser therapies is the automation of certain aspects of laser operation. This technology is now being employed in state of the art LASIK[1] eye surgery as discussed later.

Gynecologists devised laser applications beginning in 1979[6]. In particular, gynecologic laparoscopic surgery has proven to have many applications for laser-based therapies [7-9]. These therapies show great promise as alternative and situationally superior treatment modalities. This application is noteworthy because researchers have

[1] "Laser Assisted in Situ Keratomileusis refers to creating a flap in the cornea with a microkeratome and using a laser to reshape the underlying cornea". Definition provided from medical.preferredconsumer.com/laser_eye_surgeons/LASIK_glossary.html

implemented a novel user interface for controlling laser ablation[10]. This user interface will be examined in more detail later in this paper.

Laryngologists have also used laser technology to endoscopically treat a number of conditions. The general laryngeal abnormalities treated using laser therapy include: (1) airway stenosis, (2) laryngeal lesions, and (3) laryngeal carcinoma. Laser technology has also found significant application within the laryngeal surgical specialty of phonomicrosurgery. Phonomicrosurgery refers to the microsurgical techniques specifically concerned with abnormalities of the vocal fold and its related structures. Within phonomicrosurgery, laser applications are primarily either ablation or cutting techniques. Examples of vocal fold abnormalities treated with the laser include vascular abnormalities, epithelial lesions, cordotomy, arytenoidectomy, and cordectomy associated with cancer excision.

The application of laser therapy to certain phonomicrosurgical procedures and, more particularly, the user interface for controlling the laser is the focus of this paper. This paper explores the current state of the art in man-machine interfaces for laser surgery in general. This analysis is then used as the basis for a new research proposal specifically focused on user interfaces for laser phonomicrosurgery. Before beginning this analysis, the following brief description of the anatomy and relevant background information related to phonomicrosurgery is provided.

II. THE LARYNX

The larynx is composed of a skeleton, musculature, and a mucosal lining. The details of the laryngeal anatomy are found in the following figure:

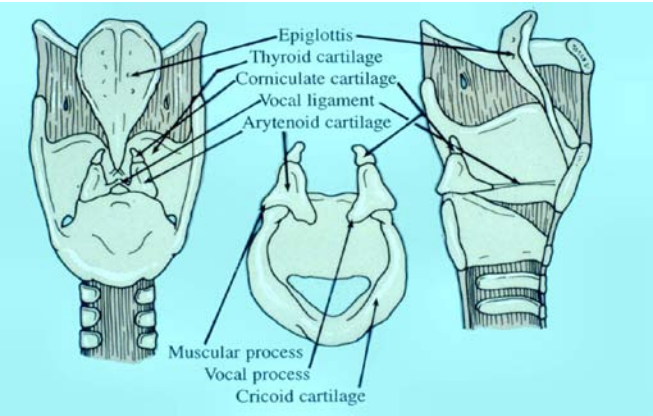


Fig. 1. Anatomy of the Larynx[11]

The cartilaginous skeleton is made up of the thyroid, cricoid, and arytenoid cartilages. This structure contains the vocal folds and is connected to the head and neck tissues by means of the extrinsic musculature. The vocal folds vary their shape, stiffness, and relative edge position under the control of the intrinsic laryngeal musculature.

To produce voice we must have airflow, sound generation and articulation. The lungs generate airflow and the vibration of the vocal folds produces sound as air moves past them. The sound generated by the vocal folds is a spectrally complex signal consisting of a strong fundamental pitch and a broad range of its even and odd

harmonics. In producing speech this broad harmonic spectrum is spectrally shaped to produce various voice sounds through the process of articulation. The voice is articulated by means of changing the shape and position of the oral cavity, specifically the tongue, lips, palate, and pharynx. In general the system functions very much like an organ pipe in which a reed (vocal folds) vibrates in response to air flow and excites a tuned pipe. In the case of the voice, the pitch of the reed and the shape of the pipe are changed dynamically to produce the appropriate voice sounds. Fig. 2 depicts the generation of sound via the vibration of the mucosal edges of the vocal folds.

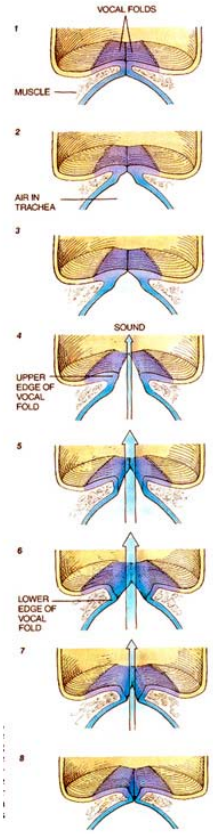


Fig. 2. Sound production[12]

When the vocal folds are compromised by undesirable tissue growths (i.e. tumors, cysts, polyps, etc.) then the voice is adversely affected.

III. OVERVIEW OF PHONOMICROSURGERY

Deglutition (swallowing), respiration (breathing), and phonation (voice production) all involve the larynx. As a result of this multi-functional nature there is a broad spectrum of disorders associated with the larynx. Not unexpectedly this leads

directly to a vast array of surgical remedies, many of which are not specific to phonation. The focus of this paper is the specialty of phonomicrosurgery [13]. Phonomicrosurgery is the art and science of manipulating the vibratory elements of the larynx in order to restore voice function. For the purposes of this paper, we will consider excisional techniques as opposed to procedures not focused on the removal of tissue. In each case of interest the goal is excision of a pathological tissue mass, however, in the first case the excision concerns a benign mass whereas in the second case the excision is of a cancerous lesion. The clinical standards for success in each case are different as are the operative approaches and procedures. The goal in cancer excision is complete removal of the malignancy while accepting that accomplishment of this objective may create vocal deficits because tissue is removed until an acceptable surgical margin is achieved. In the case of excising a benign tissue mass the success is gauged by the degree to which normal voice function may be restored. Here the surgeon is specifically concerned with minimizing trauma to the surrounding tissue while removing the objectionable pathology.

The phonomicrosurgical procedures described in the following sections focus on the correction of abnormalities. Prior to addressing the abnormal it is often useful to examine normality in order to establish a reference point. Toward that end Figure 3 and Figure 4 depict normal vocal fold anatomy. Note that in normal vocal fold anatomy the medial edges of the vocal folds are straight and thus enable them to smoothly touch

during phonation. The surface texture is smooth and shiny as opposed to the rough and irregular nature of certain pathologies discussed later.

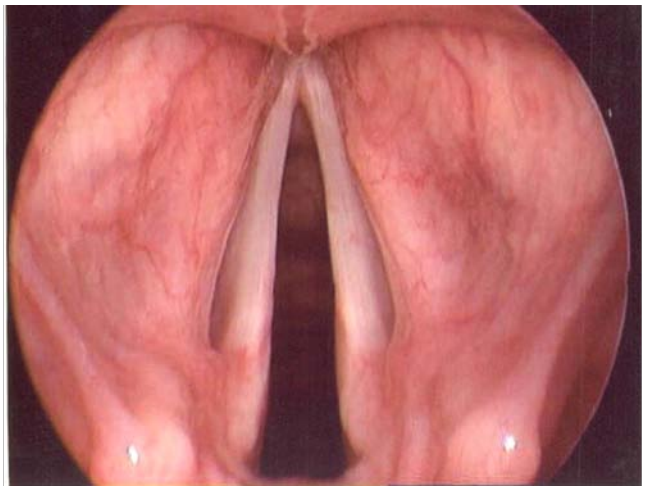


Fig. 3. Endoscopic view of normal larynx[11]

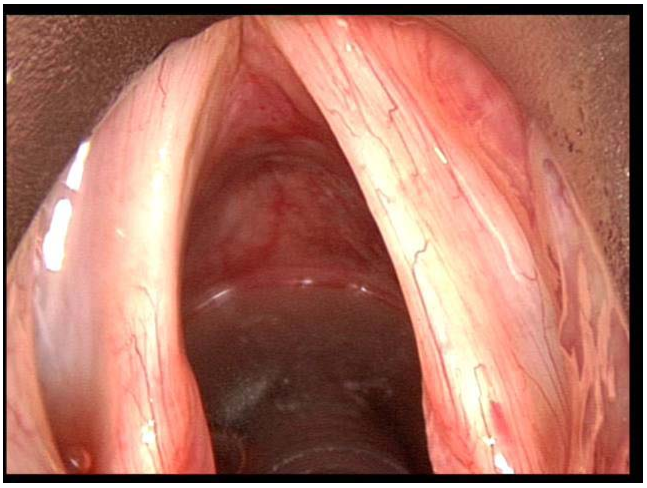


Fig. 4. Surgical view of normal vocal folds using direct laryngoscopy[11]

The first procedure of interest involves the removal of a benign cyst from the vocal fold(s). The cyst causes misalignment of the edges of the vocal folds during speech and thus adversely affects voice quality. An example of a vocal fold cyst is shown in Fig. 5 below:

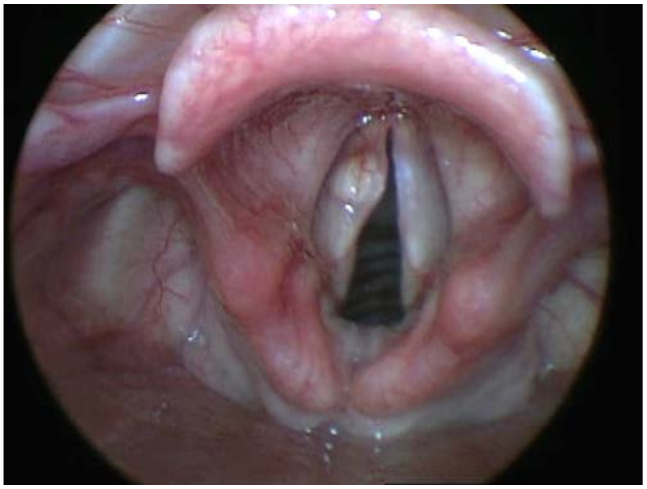


Fig. 5. Endoscopic view of benign vocal fold cyst[11]

In Fig. 5 a pronounced cyst is observed on the left vocal fold, clearly distorting the straight medial edge present in a normal vocal fold. This individual will have some symptoms of dysphonia. Note the difference in view presented in this image versus that of Figure 3. The images in Figures 3 and 5 were acquired in clinic using endoscopy to visualize the larynx as opposed to Figure 4 that was obtained via direct laryngoscopy.

Another perspective of a benign cyst is presented in Fig. 6 obtained during endoscopic surgery.

Fig. 6. Endoscopic view of benign vocal fold cyst[11]

In Fig. 6 the large cyst(s) present is clearly seen in the right vocal fold, causing easily observable deformation of the vocal fold edge.

The second procedure of interest involves the removal of a malignancy from the vocal fold(s). The clinical goal for this procedure is complete excision of the cancer. In achieving this goal voice quality is often adversely affected because of the quantity of tissue that must be removed as well as its type and location. These issues are readily observed in the example of an advanced cancerous lesion show below in Fig. 7.

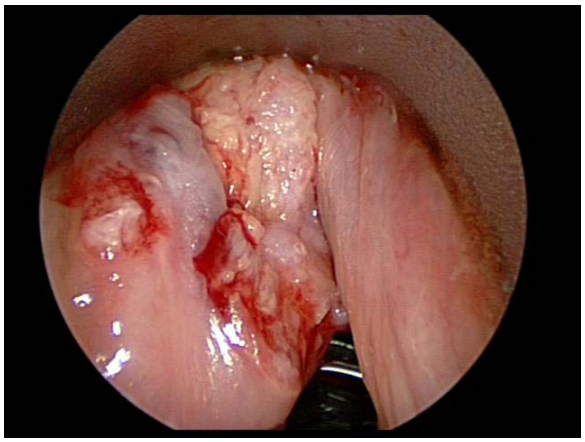


Fig. 7. Cancerous lesion of the vocal fold viewed via direct laryngoscopy[11]

In Fig. 7 the cancerous lesion on the left vocal fold is clearly observed. The image also highlights the challenges that confront the clinician with regard to visualization of the tissue mass to be excised, as well as defining the boundaries between normal and diseased tissue. In this case the tumor was inoperable endoscopically primarily because it extends in the anterior direction beyond the perimeter exposed in the laryngoscope. Typically operative feasibility requires that a safe margin around the perimeter of a tumor excision be visualized by the surgeon.

IV. SURGICAL CHALLENGES

Vocal fold surgery poses many challenges to the surgeon. First is the size of the vocal fold itself. The human adult vocal fold typically ranges in size from 17-21mm in males and 11-15mm in females[14]. Next, visualization of benign and malignant tissue

within the operative field is nontrivial. The surgeon must examine the tissue of interest using different angular views provided by endoscopes inserted into a laryngoscope. The laryngoscope is a thin walled cylindrical tube measuring approximately 7 inches long and 0.75 inches in diameter that has been placed into the oral cavity or pharynx of the anesthetized patient. Typically fiberoptic telescopes having a 0 degree or "straight on" view and a 70 degree view are employed in the examination. After mentally composing a three dimensional picture of the operative field based upon these endoscopic views, the surgeon makes decisions about the boundaries of the tissue to be excised and determines the appropriate incision locations and depths. Using relatively long operative tools the surgeon is constrained in his/her movements by the dimensions of the laryngoscope. An illustrative example of the laryngoscope and its positioning is provided in Figure 8.

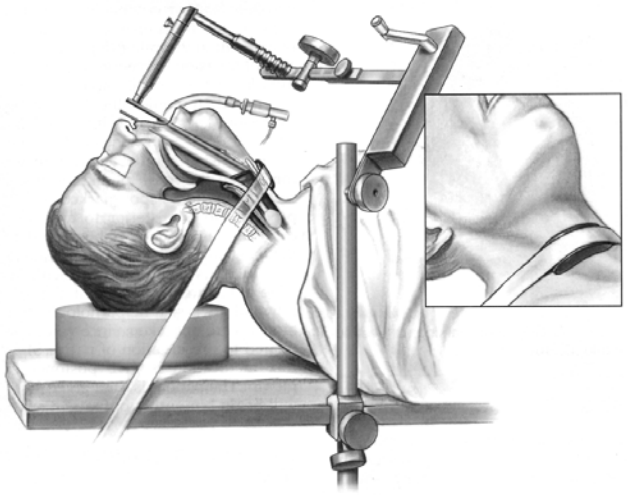


Fig. 8. Boston University suspension gallows [15]

Jako[16] is widely credited with the development of the first microlaryngeal surgical techniques in 1962. Ultimately he went on to expand the accepted range of surgical techniques to include the use of surgical lasers in phonomicrosurgery[17-19]. As in other medical specialties, phonomicrosurgery employs a number of different laser technologies. The CO_2, KTP (Potassium Titanyl Phosphate), and Nd:YAG lasers are all used in phonomicrosurgery. The specific surgical application often determines the selection of laser technology[20]. For example, the CO_2 laser has been the workhorse for laryngeal surgery for the last three decades because it provides superior performance in cutting or ablative applications. CO_2 surgical laser applications include airway stenosis, vascular lesions, benign laryngeal lesions, and laryngeal carcinoma[21].

V. LASER AIMING METHODOLOGIES

Today's surgical lasers are employed with optics tailored to the 400mm operative distance required in phonomicrosurgery. This distance is one of the challenges facing developers of laser aiming devices. Another equally significant issue is the fact that the typical CO_2 laser operates on a wavelength of approximately 10.6 μm (λ = 10.6 μm) which is outside of the visible light spectrum. Thus it is necessary to have a visible means of registering and aiming the CO_2 laser beam during operative use. This aiming beam is often a coaxial Helium Neon (HeNe) laser.

There are two prevalent methods used for remotely aiming CO_2 surgical lasers. The dominant of these two methods is the manual micromanipulator[11]. A micromanipulator is a mechanical device that orientates an optical mirror system to control the aiming of the laser beam to ensure accurate localization on the tissue of interest. The other method is the scanner. This method also employs optical manipulation to aim the laser but the optics move under computer control.

An example of a micromanipulator is shown in Fig. 9. This device is affixed to the base of the operating microscope and the surgeon visualizes the operative field directly through the specialized optics of the micromanipulator. These optics allow the passage of visible light while simultaneously allowing the reflection of the HeNe aiming beam and coincident CO_2 laser beam. The aiming point is adjusted by mechanically manipulating the small joystick and observing the position of the HeNe laser aiming

beam. The CO_2 laser beam is coupled into the system by means of a flexible waveguide that attaches to the top of the micromanipulator.

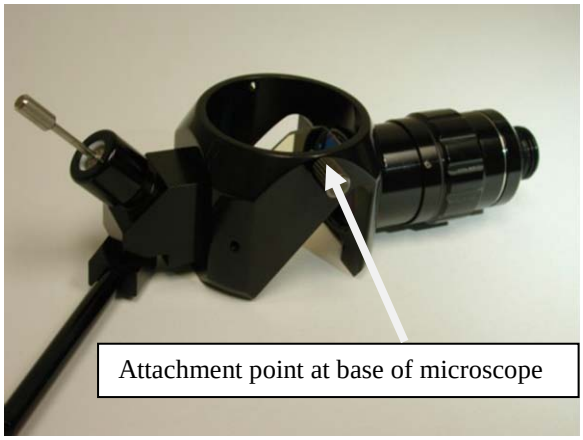


Fig. 9. Mechanical micromanipulator[11]

A system level perspective is provided by Fig. 10.

Fig. 10. Surgical microscope with attached micromanipulator and CO_2 laser[11]

The actual operating theater is depicted in following Fig. 11 and clearly shows the placement of the operating microscope. In Fig. 10 the position of the micromanipulator is clearly identified at the base of the microscope. The surgical laser itself is located out of the immediate area and is coupled to the micromanipulator by means of an optical waveguide.

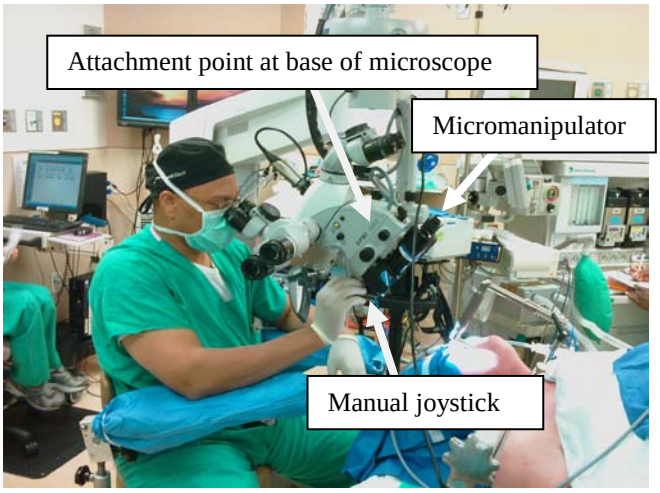


Fig. 11. Typical operative setup for laser phonomicrosurgery[11]

The two most predominant types of micromanipulators are: (1) reflective, and (2) refractive, each named for the type of optics employed. The two design approaches are depicted in Fig. 12:

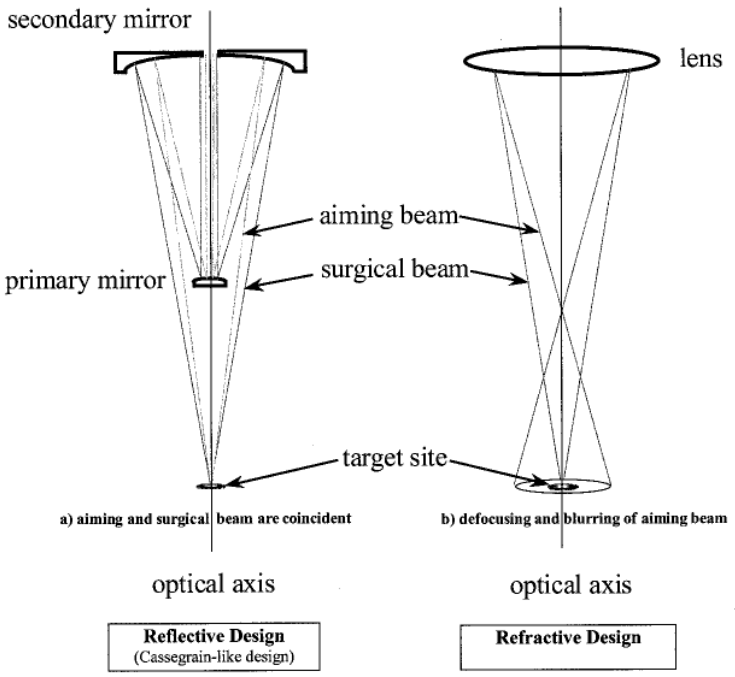


Fig. 12. Schematic diagrams of reflective and refractive micromanipulators[22]

Refractive optics are the most commonly used optics for aiming beams in phonomicrosurgery. In such a design refractive lenses are assembled to refract both the aiming and surgical beams onto the target tissue. This system has several disadvantages: (1) since the aiming and surgical beams are refracted differently, the two beams are not necessarily coincident. This may lead to unintended ablation of tissue, (2) this type of system requires periodic alignment, but this alignment is constrained by the fact that the

refractive properties of the lenses are a function of wavelength. Newer micromanipulator designs employ reflective optics, which have the advantage of eliminating the wavelength dependency for the focusing operation, and ensure that the laser beams are coincident within the entire spectra of interest. Using reflective optics technology advances the performance of the system to the point of only being limited by the surgeon's skill in operating the micromanipulator, which requires significant skill and practice. Hand tremor, the small nature of the target area, and the 400mm operative distance which amplifies small errors in manipulator adjustment all combine to make it very difficult for novice clinicians to guide the laser as accurately as experienced surgeons.

The term scanner refers to the use of optics to control the direction or aiming of a surgical laser. Much of the body of work in this area has been for ophthalmologic applications and there are some general characteristics within ophthalmologic surgery that are also applicable in other surgical situations. For example, one major consideration is the idea of an exclusion zone or prohibited area for firing the laser. Any time the aiming point of the laser falls within this exclusion zone the laser firing mechanism is inhibited. In ophthalmologic surgery one such zone is the fovea. In the case of phonomicrosurgery the airway is the exclusion zone in order to minimize the risk of an intraoperative airway fire or injury. The ability to discern exclusion zones is fundamentally a tracking problem. The system must at all times maintain awareness of

the aiming position of the laser relative to any exclusion zones. For example, Kuruganti describes an interesting system employing Micro-ElectroMechanical Systems (MEMS) devices as a core component of a graphical user interface (GUI) for laser surgery [23]. One of the more significant outcomes of his work was proving the suitability of the MEMS microactuator device in this application. The GUI proposed by [23], and its role in aiding the clinician, is discussed more fully later in this paper.

VI. USER INTERFACES

It has become common in many applications to refer to user interfaces involving computer software as the human-computer interaction or HCI. Most of us are trained to think of HCI in terms of keyboards, pointing devices such as mice and trackballs, and joysticks. As a result these familiar devices are the foundation for most new HCI initiatives. HCI in medicine finds many potential applications including computer-based training for surgery and computer assisted surgery, among many others.

Considerable effort has been devoted to creating computer-based surgical training systems. These systems are intended to allow trainees to perform complex procedures and to then evaluate their performances against key clinical metrics. Typically the "gold standard" for a given procedure is the benchmark clinical metrics derived from an expert surgeon performing the procedure. The trainee's performance is then compared to these benchmarks. Some of these computer-based training initiatives have focused upon

virtual reality systems as a technology platform. As Arnold and Farrell [24] report in their critical literature review on this subject, this technology has delivered little of practical value to date to practicing surgeons. Separately Arnold has also analyzed the difficulty of performing a fine motor skill in a virtual environment (VE) [25]. He concluded that it is much more difficult to perform a fine motor skill in a VE as compared to a real environment. This finding may be the underlying cause for the lack of adoption of VE technology in surgery and surgical training today. Thus it appears that pursuing a VE type of enhanced user interface for laser phonomicrosurgery is not the best path forward at this time.

In addition to computer based surgical training a significant amount of advanced HCI work has been done specifically for LASIK operator interfaces in ophthalmology. The main challenges in ophthalmic surgery relate to the uncontrolled movement of the operative field and to executing precise, repetitive operations within that environment. Thus, much of this body of work has focused on the development of automated retinal tracking systems [5, 26] to ensure accurate aiming of the laser beam prior to activating the laser. Kuruganti's work is of interest in this area as he also developed a GUI for his system. The control screen of this GUI is depicted in the following figure:

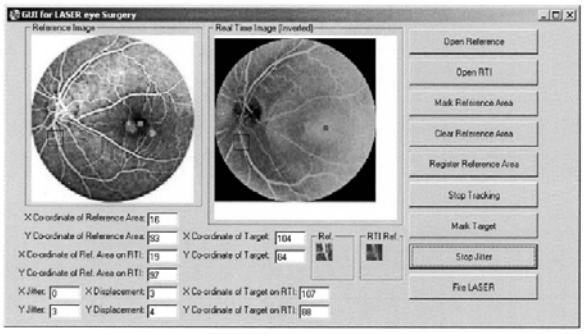


Fig. 13. GUI for laser eye surgery system[23]

The image on the left of Figure 13 is a reference image while the right side figure is a real time image. The real time image is registered to the reference image in such a way as to allow the control software to determine whether or not the laser is in a safe firing zone. That is, the landmarks established in the reference image enable the system to identify exclusion zones and inhibit laser firing while the laser is aimed into these areas.

The classic mechanical micromanipulator is an effective surgical laser aiming mechanism in the hands of a skilled and experienced clinician. Unfortunately it is not an easy device to master and typically requires significant clinical experience to employ effectively. This training period often takes years to complete since many different clinical situations must be experienced for thorough training to be conducted.

The laser scanner addresses some of the challenges inherent in the micromanipulator. Automating the optics movement affords the opportunity to mitigate some of the pitfalls of the manual micromanipulator. These issues include hand tremor

as well as imprecise or inappropriate target selection. However, few advances in this area have been reported. In fact, to date commercially available scanner systems such as the AcuBlade™[2] focus on reducing operative setup times. This was done via a menu-based approach that allowed a clinician to select the general surgical parameters desired for a particular procedure. This reduces setup time since the laser system is automatically configured based upon selections from a pre-ordained menu. While more convenient than manually configuring the laser system, this capability does not enhance the actual user interface in the performance of surgical tasks. Toward this end, however, this system does offer the capability to make incision paths or ablate tissue via scanning. Both of these operations are closely controlled and guided by the clinician.

An alternative approach for a user interface is found in work done by Tang, Van Brussel, Sloten, Reynaerts, and Koninckx [10]. Tang and his associates developed a novel writing-based interface for defining an excision path. The essential premise of this research was that human handwriting is a more accurate approach to surgical laser control than traditional manual laser laparoscopy. In this system the surgeon uses a tablet PC to draw a desired excision perimeter and this outline is then executed by a robotically controlled laser. System performance of their prototype was effective, having a tracking error of 1mm while operating at an ablation speed of 20mm/sec. A high level block diagram of the system is shown in Fig. 14.

[2] Lumenis Ltd, Santa Clara, CA

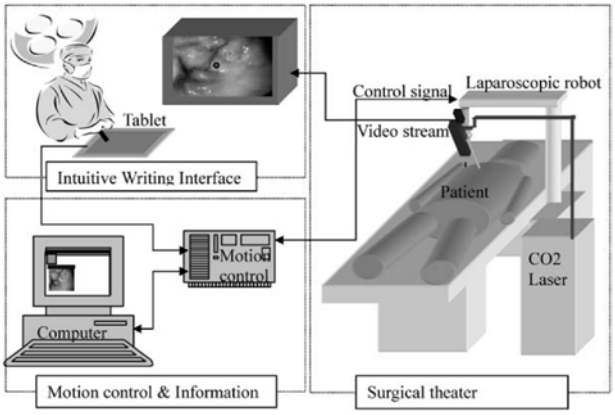


Fig. 14. Robot assisted laser surgical system[10]

Traditionally in any robotic slave system, particularly in a robotic surgical tool, generally there is also the user desire for tactile feedback. Rizun and Sutherland have developed the concept of haptic feedback for laser systems [27, 28]. The focus of their work improved the human computer interaction (HCI) by providing the operator with the ability to sense the three dimensional nature of tissue to be ablated. The goal was to allow the operator to control the ablative force brought to bear by the laser by means of haptic force feedback. The utility of this capability is likely to be of limited value in laser phonomicrosurgery. In phonomicrosurgery most of the feedback cues that a surgeon receives are visual rather than tactile, particularly when the laser is employed rather than "cold steel"[3] surgical instruments.

[3] "Cold Steel" refers to the use of traditional steel surgical instruments

There appears to be very little in the literature pertaining to semi and fully automated surgical laser user interfaces. Thus far the commercial thrust appears to focus on reducing the workload associated with the pre-operative laser setup, as opposed to enhancing the capability of the overall operative system. A significant advancement in the design and use of interactive user interfaces will automate certain tasks within a given set of operative procedures. For example, it will be possible for an enhanced system to allow the clinician to define an ablation path and memorize this path. The path could then be automatically cut by the laser in either "dot by dot" or continuous mode. The classic errors related to maintaining the desired excision path and "off target" laser shots will be reduced, if not eliminated, by implementing this type of semi-autonomous system operation. It may also be valuable to develop a definition of safe firing zones. The safe target area in phonomicrosurgery for vocal fold procedures is typically within an area measuring approximately 17mm long by 5mm wide. Preventing laser firing when the aiming point is outside of the safe firing zone would further enhance patient safety.

Although intended for a wholly different research purpose, the work published by Reinisch, Mendenhall, Charous, and Ossoff [29] is of interest in this area. Using the free electron laser at Vanderbilt University, Reinisch and his team created a Computer-assisted Surgical Techniques (CAST) system to enable computer controlled ablation. The system employs a basic GUI running on an Apple Macintosh™ computer that in turn controls a laser scanner mechanism that effects laser beam aiming. Reinisch and his

colleagues applied this technology to otolaryngology but were specifically interested in understanding the mechanisms associated with charring of temporal bone as a function of various laser operating parameters. The laser user interface that they developed was an enabling technology to perform this experiment and was thus not a primary focus of their research. In this system the user selects aiming points within a screen shot of the target area. A sample control screen from their interface is shown in Fig. 15.

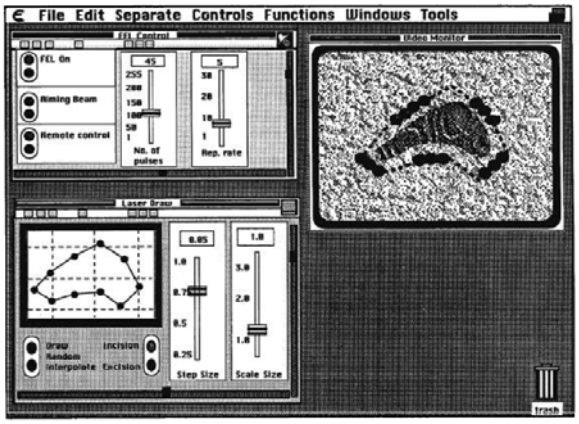


Fig. 15. Display screen for Vanderbilt CAST system[29]

The CAST system allows the operator to define an excision path for the laser and either ablate everything within the confines of the outline or proceed in a conventional "dot by dot" approach. The system can fire the laser directly or relinquish firing control to the operator via the traditional foot pedal switch. Reinisch and his coauthors report

that the scanner-based control of the laser aiming afforded excellent control as well as the ability to accurately retrace a given outline repetitiously. There was no discussion of safety features nor of system failure modes. In particular there was no mention of how an exclusion zone might be implemented. As a result, if the system were to malfunction in a clinical setting there would be a risk of patient injury. Additionally, should the clinician intraoperatively opt to replace the device with the traditional micromanipulator this would likely incur a significant amount of downtime in order to perform the changeout. This downtime would increase the general anesthesia interval for the patient thereby increasing the risk of complications. These issues lead us to consider an alternative implementation of an improved surgical laser aiming HCI. Table 1 summarizes the current landscape in user interfaces for surgical laser aiming.

Table 1. Comparison of laser aiming technologies

	Mechanical micromanipulator	Accublade™	Tang et al Tablet system	Vanderbilt CAST system	New research HCI
Optics control	Mechanical	Scanner	Scanner	Scanner	Mechanical
HCI type	N/A	Touch pad	Writing tablet	Mouse	Joystick
GUI	N/A	yes	yes	yes	yes
Accuracy	+/-	N/A	+	+	++
Automation	none	limited	partial	extensive	full

It is clear that the industry has not settled on any particular standard for an HCI or optics control. There appears to be uniformity in the use of GUIs for system control while system accuracy varies by implementation. The newer offerings incorporate higher levels of automation although the focus and type of automation varies considerably. Overall the industry exhibits the characteristics one would expect with new and evolving technology.

VII. RESEARCH OBJECTIVES

We propose the design and development of an enhanced user interface for aiming surgical lasers. The primary goal will be to develop an HCI that will enable more rapid user training and improved aiming accuracy. Although the initial application will be phonomicrosurgical procedures it is anticipated that the enhanced user interface will find broader applications beyond this surgical specialty. Potential benefits of such a user interface would include reduced aiming deviations due to hand tremor, the memorization of exact excision paths for automated repetition, and a general improvement in the consistency of results achieved by clinicians, particularly the less experienced. Table 1 summarizes the general characteristics of the proposed new HCI.

The longer term goal of the research will be to facilitate improvement in clinical outcomes for certain laryngeal surgical procedures. This will be accomplished by

increasing the overall accuracy of general surgical laser aiming while reducing operative times through the introduction of semi-automation of certain tasks.

The proposed new HCI is envisaged as a mechatronic system that will interface to the existing manual micromanipulator. The new system will be a combination of hardware and software having the following proposed high level system architecture:

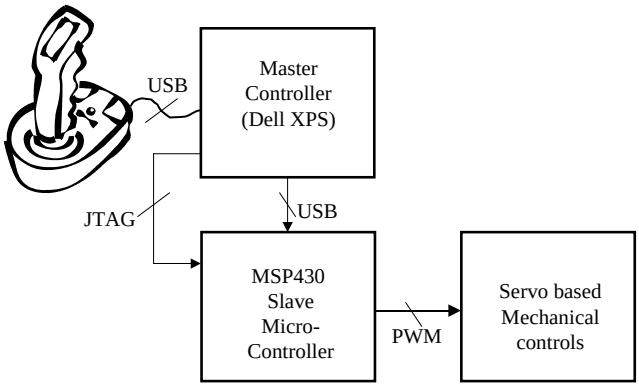


Fig. 16. Medical Robotic System architecture

The device design will be modular in nature in order to facilitate both rapid installation and removal in the event that circumstances dictate a more traditional approach to a given situation. Control functions for the system will be centralized in GUI software running on the master controller. The master control software will interpret joystick position and provide appropriate commands to the slave microcontroller to actuate the mechanics and thereby alter the surgical laser aiming point. Advanced

features, such as path memorization and playback will be implemented in the GUI software.

The overall utility of the enhanced HCI will be evaluated in trials with clinicians of various skill and experience levels. Experiments will be devised to measure task performance using the traditional manual micromanipulator as well as the new HCI. Comparisons of speed and accuracy will then be made to ascertain the benefits, if any, of the new HCI.

REFERENCES

[1] T. H. Maiman, "Stimulated Optical Radiation in Ruby," *Nature,* vol. 187, pp. 493-494, 1960.

[2] J. L. Bromberg, *The laser in America, 1950-1970*. Cambridge, Mass.: MIT Press, 1991.

[3] T. G. Polanyi, Bredemei.Hc, and T. W. Davis, "Co2 Laser for Surgical Research," *Medical & Biological Engineering,* vol. 8, pp. 429-&, 1970.

[4] E. L. Tanzi, J. R. Lupton, and T. S. Alster, "Lasers in dermatology: four decades of progress," *J Am Acad Dermatol,* vol. 49, pp. 1-31; quiz 31-4, Jul 2003.

[5] M. S. Markow, Y. Yang, A. J. Welch, H. G. Rylander, and W. S. Weinberg, "An Automated Laser System for Eye Surgery," *Ieee Engineering in Medicine and Biology Magazine,* vol. 8, pp. 24-29, Dec 1989.

[6] M. A. Bruhat, Mage, G., and Manhes, H., "Use of the CO2 laser via laparoscopy," in *Proceedings of the Third Conference on Laser Surgery*, Tel Aviv, Israel, 1979, pp. 274-276.

[7] H. Hibi, K. Kato, K. Mitsui, T. Taki, Y. Yamada, N. Honda, and H. Fukatsu, "Endoscopic ureteral incision using the holmium:YAG laser," *Int J Urol,* vol. 8, pp. 657-61, Dec 2001.

[8] J. H. Hong, S. S. Jeon, and K. S. Lee, "Result of endoscopic ureteroureterostomy with holmium:YAG laser for complete ureteral obstruction," *J Endourol,* vol. 19, pp. 979-83, Oct 2005.

[9] E. Lopriore, G. van Wezel-Meijler, J. M. Middeldorp, M. Sueters, F. P. Vandenbussche, and F. J. Walther, "Neurodevelopmental outcome after laser therapy for twin-twin transfusion syndrome," *American Journal of Obstetrics and Gynecology*, vol. 196, p. e20, 2007.

[10] H. W. Tang, H. Van Brussel, J. V. Sloten, D. Reynaerts, and P. R. Koninckx, "Implementation of an intuitive writing interface and a laparoscopic robot for gynaecological laser assisted surgery," *Proceedings of the Institution of Mechanical Engineers Part H-Journal of Engineering in Medicine*, vol. 219, pp. 293-302, Jul 2005.

[11] R. Buckmire, M. D., 2006, p. Image provided courtesy of Dr. Buckmire.

[12] C. A. M. D. Rosen, T. P. D. Murry, and May, "The University of Pittsburgh Voice Center," University of Pittsburgh, 2006.

[13] S. M. Zeitels, *Atlas of phonomicrosurgery and other endolaryngeal procedures for benign and malignant disease*. San Diego, Calif.: Singular, 2001.

[14] D. M. Bless and J. H. Abbs, *Vocal fold physiology :contemporary research and clinical issues*. San Diego, Calif.: College-Hill Press, 1983.

[15] e. a. Hochman, "Exposure and visualization of the glottis for phonomicrosurgery," *Operative techniques in Otolaryngology - Head and Neck Surgery*, vol. 9, pp. 192-195, 1998.

[16] G. J. Jako, "Laryngoscope for Microscopic Observation, Surgery, and Photography . Development of an Instrument," *Archives of Otolaryngology*, vol. 91, pp. 196-&, 1970.

[17] D. Jako and S. Strong, "Use of Co2-Laser in Microsurgery of Larynx," *Hno*, vol. 22, pp. 122-122, 1974.

[18] G. J. Jako, "Laser Surgery of Vocal Cords - Experimental Study with Carbon-Dioxide Lasers on Dogs," *Laryngoscope*, vol. 82, pp. 2204-2216, 1972.

[19] G. J. Jako, M. S. Strong, T. G. Polanyi, and Bredemei.Hc, "Experimental Carbon-Dioxide Laser Surgery of Vocal Cords," *Eye Ear Nose and Throat Monthly*, vol. 52, pp. 171-172, 1973.

[20] R. H. Ossoff, J. A. Coleman, M. S. Courey, J. A. Duncavage, J. A. Werkhaven, and L. Reinisch, "Clinical-Applications of Lasers in Otolaryngology - Head and Neck-Surgery," *Lasers in Surgery and Medicine*, vol. 15, pp. 217-248, 1994.

[21] R. A. Buckmire, MD; Associate Professor of Otolaryngology/Head & Neck Surgery, University of North Carolina; Director, UNC Voice Center, "Lasers in Laryngology," Chapel Hill, 2006, p. 21.

[22] K. K. H. Chao, E. Cheung, W. B. Armstrong, and B. J. F. Wong, "The effect of optical design on micromanipulator spot size using CO2 laser irradiation," *Otolaryngology-Head and Neck Surgery*, vol. 126, pp. 593-597, Jun 2002.

[23] V. K. Kuruganti, "Graphical user interface for improved laser eye surgery," in *Department of Electrical and Computer Engineering*. vol. Master of Science in Computer Engineering: New Jersey Institute of Technology, 2003, p. 65.

[24] P. Arnold and M. J. Farrell, "Can virtual reality be used to measure and train surgical skills?," *Ergonomics*, vol. 45, pp. 362-379, Apr 2002.

[25] P. Arnold, M. J. Farrell, S. Pettifer, and A. J. West, "Performance of a skilled motor task in virtual and real environments," *Ergonomics*, vol. 45, pp. 348-361, Apr 2002.

[26] R. Marmulla, T. Luth, J. Muhling, and S. Hassfeld, "Automated laser registration in image-guided surgery: evaluation of the correlation between laser scan

resolution and navigation accuracy," *International Journal of Oral and Maxillofacial Surgery*, vol. 33, pp. 642-648, Oct 2004.

[27] P. Rizun and G. Sutherland, "Tactile feedback laser system with applications to robotic surgery," 2005, pp. 426-431.

[28] P. R. Rizun and G. R. Sutherland, "Surgical laser augmented with haptic feedback and visible trajectory," 2005, pp. 257-260.

[29] L. Reinisch, M. Mendenhall, S. Charous, and R. H. Ossoff, "Computer-Assisted Surgical Techniques Using the Vanderbilt Free-Electron Laser," *Laryngoscope*, vol. 104, pp. 1323-1329, Nov 1994.

CHAPTER 2. Design of a Medical Robotic System for Laser Phonomicrosurgery

Abstract—The paper chronicles the design, implementation, and testing of a medical robotic system intended for automating laser phonomicrosurgery. Two designs were built and tested. The first part of this paper catalogs why the original design failed because of controllability issues. The second design was based on a precision electro-mechanical device controlled by computer. Minor design adjustments to this second system were made following a series of clinical trials and the new design performed very well. Clinical evaluation trials are continuing to assess the ease of use, accuracy, and the overall potential utility of this system as an alternative to the classic mechanical micromanipulator for laser phonomicrosurgery.

I. INTRODUCTION

The dominant surgical laser user interface for laser phonomicrosurgery is the manual micromanipulator depicted in Fig. 1. Investigation of the field concluded that there existed a significant opportunity to improve the man machine interface employed for aiming the surgical laser and for robotizing laser phonomicrosurgery[1]. The first step explored how best to improve the interface, and this paper describes that process. A new medical robotic system design is presented, one that includes an enhanced Human Computer Interface (HCI) for use in laser phonomicrosurgery. The many design challenges that were faced in developing the system are analyzed.

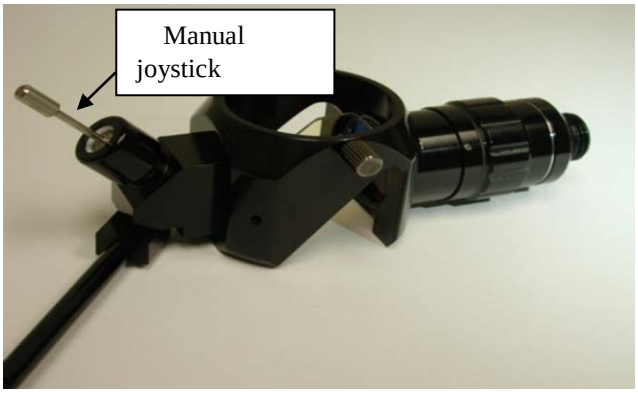


Fig. 1. Manual Micromanipulator[2]

Fig. 1 depicts the manual micromanipulator and shows the location of the joystick used to move the optics of the device. The ergonomics of the manual micromanipulator require the clinician to grasp the manual joystick and move it using the fingers of either hand. No support is provided for the wrist and the forearm support is suboptimal. This

increases the difficulty of making small, precise movements and then replicating them where repeated patterns are desired. These challenges are captured in Fig. 2.

Fig. 2. Laser phonomicrosurgery operative setup[2]

After considering the operative environment and the differing requirements for various surgical procedures it was decided that a medical robotic approach to the problem was appropriate.

The basic goal of the medical robotic system is to provide the clinician with a means of aiming a surgical laser such that the limitations of the current technology, described above, are removed. A number of design approaches were evaluated in the process of determining the system architecture for a functional prototype device [3-5]. As described in the comparative analysis of Giallo, et al [1], it was desired to achieve measurable advances in each of the following design areas:

- Graphical User Interface (GUI)
- Accuracy and repeatability
- Automation

II. MEDICAL ROBOTIC SYSTEM I DESIGN AND IMPLEMENTATION

The first system designed utilized a scanner-based approach. A scanner offers some advantages over more complex mechatronic implementations. Typically there are few moving parts and the mechanical implementation is simplified as a result. The associated control circuitry likewise is not onerous and thus leads to a relatively straightforward implementation. A block diagram of the proposed system is shown in Fig. 3.

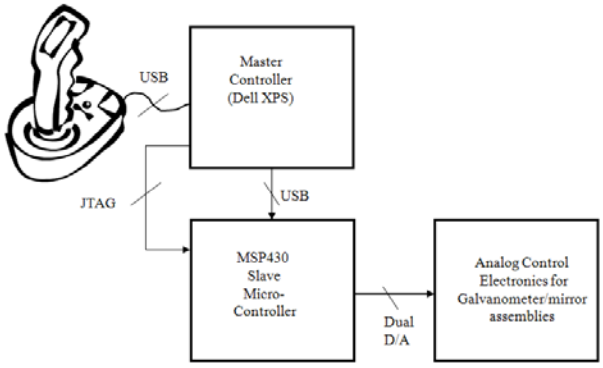


Fig. 3. Initial Prototype system diagram

For cost reasons, and to allow system rapid prototyping, it was decided to utilize off the shelf components to construct the proof of concept prototype. The essence of any scanner is a moveable mirror. A low cost mirror/galvanometer combination was identified and two such units were purchased to implement the x and y axis controls. The galvanometer responds to a changing analog input signal by rotating its output shaft. The characteristics of the analog input signal determine the magnitude and direction of the output shaft motion. The interface circuit of Fig. 4 was designed to provide this control function.

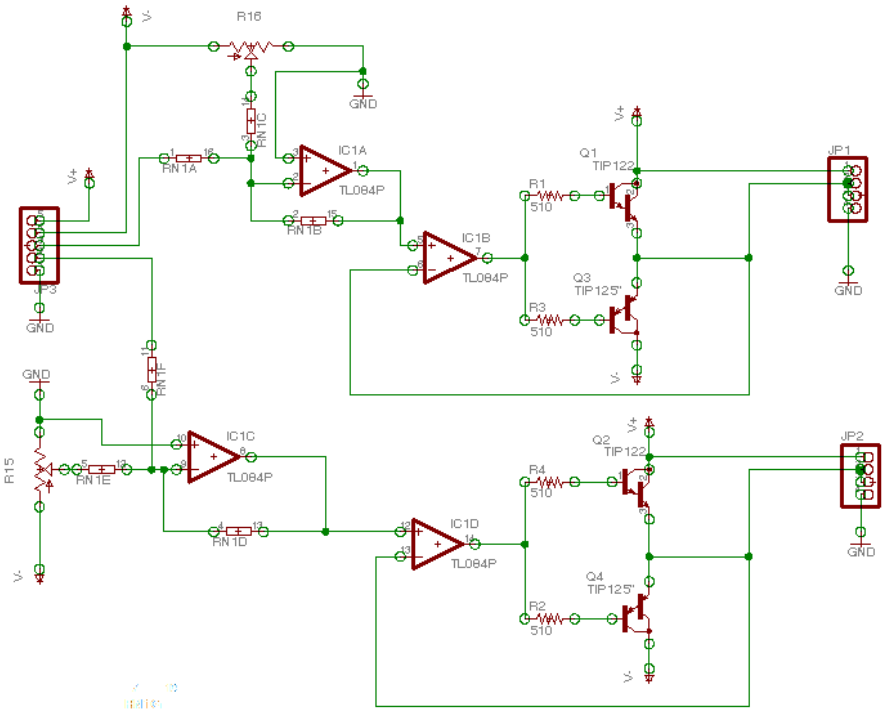


Fig. 4. Galvanometer control circuit

The galvanometer control circuit of Fig. 4 is essentially a level shifting buffer circuit combined with a voltage follower and output drive stage for the galvanometers. The high current drive required by the galvanometers is provided by means of a push-pull Darlington transistor pair. This is necessary since the operational amplifiers are not capable of sourcing the required current. These current requirements also make it necessary to provide substantial heat sinks for the drive transistors during the continuous operations required in this application. The circuit itself was implemented on a double-sided printed circuit board in order to accommodate physical footprint constraints and ensure a relatively noise free design. The operational amplifiers used in the circuit are the Texas Instruments TL084 which combines four devices in a single 14 pin DIP. The resistors are implemented using a combination of discrete components and a monolithic array contained in another 14 pin DIP. The implementation of Fig. 4 is shown in Fig. 5.

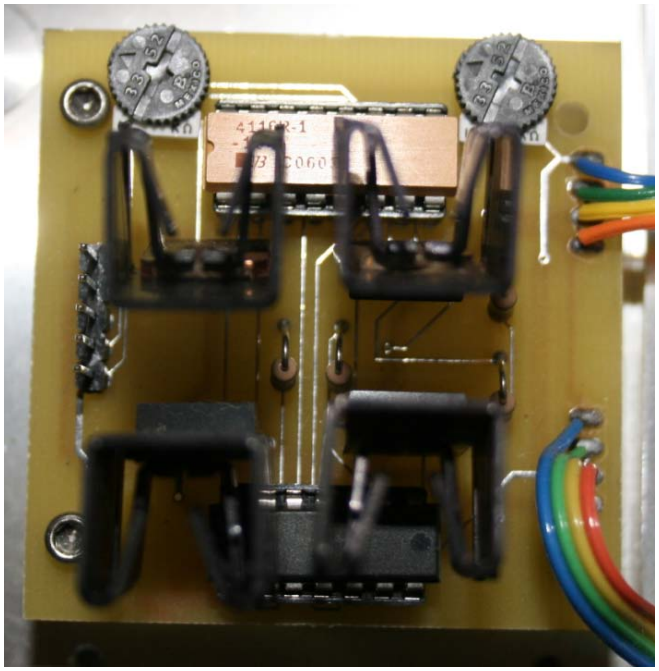


Fig. 5. Galvanometer control circuit implementation

The exciting circuit is not shown but is based upon a Texas Instruments MSP430F169 microcontroller. The purpose of the microcontroller is to provide the analog input voltage to drive the front end operational amplifiers of Fig. 4. The MSP430 series of components is a versatile family of embedded controllers and provides many useful monolithic functions. In this case, the dual 12-bit digital-to-analog (D/A) channels on the microcontroller were used to source the analog input signals for the x and y axis galvanometers respectively.

The next challenge of a scanner design dealt with mounting the mirrors as accurately as possible. To accomplish this task a special fixture was machined to hold

the galvanometer/mirror assemblies at a precise right angle. The units were secured in the fixture by means of soft-tipped set screws. This arrangement also allowed for slight position adjustments to be made in order to optimize the mirror locations relative to one another. Further adjustments are possible as a function of the fixture mounting apparatus itself. The completed fixture is shown in Fig. 6. The mounting apparatus of the galvanometer fixture enables adjustments in the x, y, and z planes as needed to ensure acceptable alignment between the mirrors and the laser output beam.

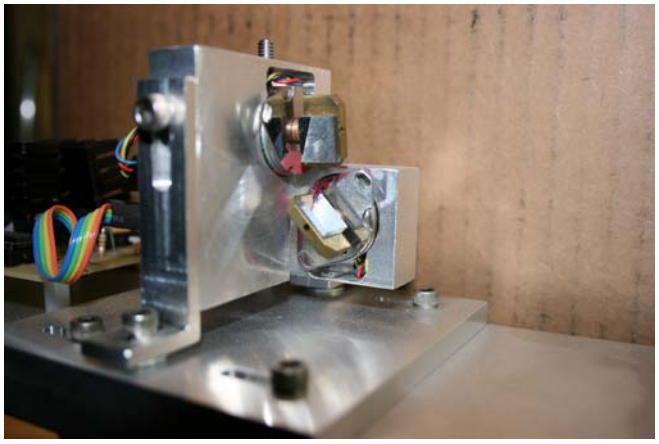


Fig. 6. Galvanometer mounting assembly

A low cost, eye safe HeNe laser source was identified and procured for initial testing purposes. This device was mounted onto the platform with the galvanometer apparatus and related control circuitry. The completed initial prototype system is shown in Fig. 7.

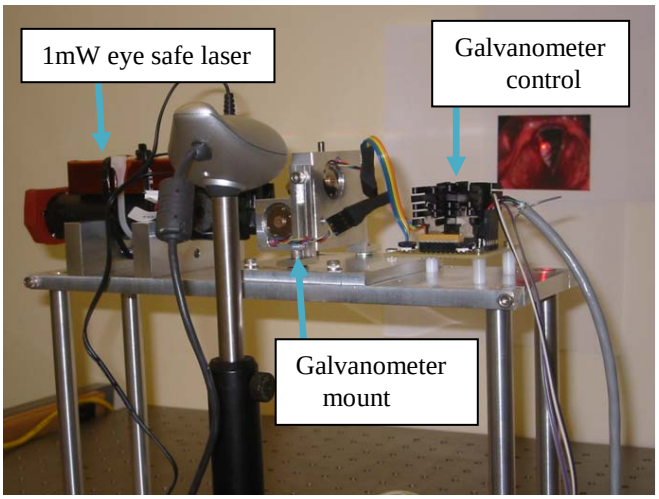


Fig. 7. Scanner prototype assembly

The plan for conducting experiments using the prototype system involved using a digital video camera to capture the HeNe aiming beam movements. The next step was to analyze these images against reference images and compute performance metrics.

Unfortunately, there were a number of challenges associated with this approach: (1) the lack of recorded hard copy media made it difficult to easily make and repeat physical performance measurements, (2) while not insurmountable, this issue required a substantial effort to resolve using image processing techniques, (3) the design objectives of improved accuracy and automation in a clinical setting were very difficult to achieve. Further, referring to Fig. 2 it is readily apparent that the operative setup imposes significant constraints: (1) there is limited physical space for placement of the apparatus. This placement must be accomplished unobtrusively in the current operative

environment, (2) the apparatus must integrate with the existing surgical laser interface. As can be observed from Fig. 2 and Fig. 7 this would be virtually impossible to achieve given the geometry of the initial prototype, (3) additionally, the scanner removes the ability of the surgeon to simultaneously visualize the operative field through the surgical microscope while the laser is activated.

For these reasons it was unfortunately necessary to abandon the initial prototype system and consider alternative design approaches.

III. MEDICAL ROBOTIC SYSTEM II DESIGN AND IMPLEMENTATION

In reconsidering the design constraints it was postulated that a medical robotic system that attaches to the existing manual micromanipulator would relieve the awkward ergonomics that exist with manual micromanipulator. Such a system could achieve this capability by remotely effectuating laser aiming. Fundamentally altering the way in which the clinician interacts with the mechanical micromanipulator was deemed necessary.

The primary challenge of such an appliance lay in its mechanical design. The design must allow: (1) simultaneous but independent movement of the micromanipulator manual joystick in both the x and y axes, (2) the independent micromanipulator movements to be smooth and proportional to the user input from the electronic joystick. This implies some type of continuously variable, possibly analog, control system. After considerable investigation into implementation alternatives, the electronic joystick itself

was examined. As stated above the electronic joystick must capture separate and independent movements in the x and y axes and transmit these movements to a controller, in this case. The joystick must convert minute mechanical motions into electrical signals and do so with reasonable fidelity. From Fig. 8 we observe that the joystick mechanics convert mechanical motions into resistance changes by means of potentiometers (see the left side of Fig. 8). These resistance changes are then used in conditioning circuits to convey the encoded electrical representations of the mechanical displacements to upstream devices.

Relating this characteristic back to the medical robotic system design we observe the need to move the tip of the micromanipulator joystick in the same fashion that the shuttle is moved in Fig. 8. Instead of potentiometers we observe that servomotors and pushrods could be used to move the gimbal and thus the shuttle in our new design.

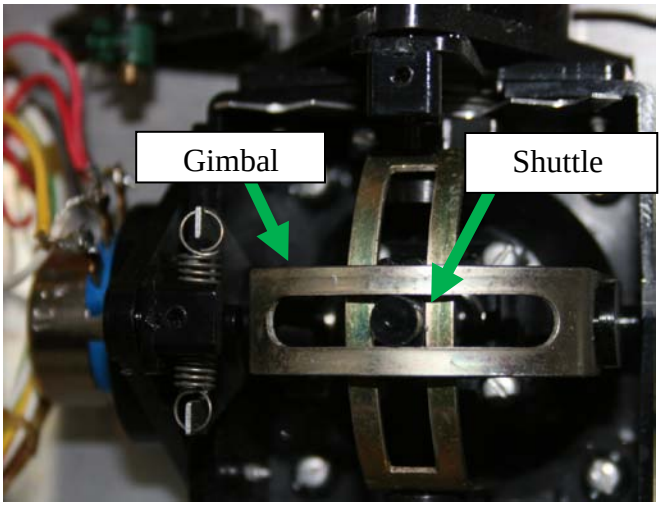


Fig. 8. Mechanics of an electronic joystick[4]

Prior to commencing the development of the second prototype it was deemed prudent to first design and model the mechanical aspects of the device. Toward this end it is first necessary to examine the required motions present in the micromanipulator. As previously noted, displacements in the x and y planes and the corresponding changes in laser aiming point are accomplished independently. That is, it is possible to move in either plane alone or in both planes simultaneously. Strictly speaking, the tip of the mechanical micromanipulator joystick is a point traversing a hemispherically-shaped surface. However, for very small movements, as are the case for typical surgery, it is hypothesized that we may assume the motion of the tip of the joystick to be linear and located in a traditional x-y coordinate plane. This assumption is the basis of the mathematical model that was developed and will be discussed in this paper.

[4] GDA-1205 8 channel R/C transmitter, Heath Corporation, Benton Harbor Michigan

The manual joystick of the classic micromanipulator shown in Fig. 1 is analogous to a pendulum. More specifically, it is possible to model the behavior of the manual joystick by employing an inverted pendulum construct. The inverted pendulum model has been well studied and documented [6]. The inverted pendulum is classically described as being mounted to a moving cart, however, in our application the mounting point is a fixed fulcrum. For our purposes it is possible to simplify some of the classic model attributes. The point mass, m, described in [6] is very small and may be ignored as a result. The basic equation of motion we wish to model is the displacement of the end of the pendulum, at m, in relation to the angle theta, θ. This displacement corresponds to the change in the laser aiming point. The nature of phonomicrosurgery requires that small, precise movements of the laser be made[1] and this fact aids the methodology employed in this analysis. Since the displacement of the end of the pendulum is small, it is assumed that the displacement is essentially a linear, planar movement. The result is the right triangle geometry shown in Fig. 9.

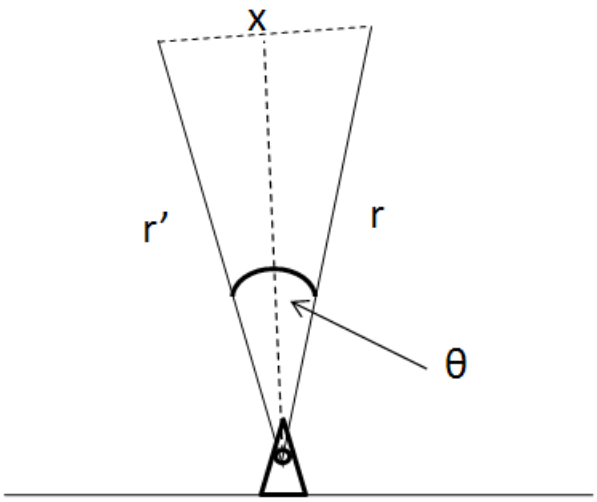


Fig. 9. Simple movement model

Fig. 9 models the movement of the manual joystick, from an initial position r, to a new position r'. The displacement x corresponds to the movement of the laser aiming point by means of a related angular displacement of the mirror assembly of the mechanical micromanipulator. The magnitude of x is described by the following trigonometric relationship:

$$x = 2r \sin\left(\frac{\theta}{2}\right) \tag{1}$$

The maximum total deflection of the manual joystick is 42 degrees, or 21 degrees in either the positive/negative x and y axis directions from a neutral center position. This movement model is easily extensible to the y plane using a second angle and similar methodology. Together, this provides us with a means of estimating the movement of the

manual joystick as a function of the angular displacement provided by the servomotors. These movements in turn cause deflections of the mirror assembly in the x and y coordinate planes. The deflections cause changes in the laser aiming point.

To model the mirror assembly deflections turn again to Fig. 9. In addition to modeling the movement of the mechanical joystick, the figure also provides a model for determining the laser aiming point. When the mechanical joystick is actuated, the movement results in some angular displacement θ per Fig. 9. This causes the mirror assembly to deflect in the x and/or y planes. Originally it was thought that the displacement of the mirror assembly was related to the displacement of the manual joystick by a simple linear scaling factor. Thus the original model for the angular displacement of the mirror assembly (defined as α for reference purposes) is related to the angle θ by a scaling factor (defined as μ for reference purposes). Unlike the manual joystick, however, the value of α (this α is substituted for θ in Fig. 9 and equation (1)) for the mirror assembly varies between 0 degrees and 3.2 degrees.

$$\mu = \theta/\alpha \tag{2}$$

By measuring the mirror deflection distance (X in Fig. 9) from a neutral point and using the fixed mirror width (r in Fig. 9) we can employ the same trigonometric relationship to determine α and the corresponding change in the x axis laser aiming point. For laser phonomicrosurgery the operative distance is 400mm so r must be adjusted accordingly. A similar approach is used to determine the change in the y axis laser

aiming point. When computing the change in the y axis aiming point the longitudinal axis of the mirror is the relevant dimension. Thus the reference dimension is the length of the mirror rather than its width. Taken together the two model components provide a means of relating a change in the position of the manual joystick to a change in the laser aiming point. Mechanical non-linearities, dead zones and gear backlash in the servomotors were not incorporated into the analysis because the basic objective of the model was to validate that adequate range of movement existed in the mechanics of the system. It was discovered during model validation that mechanical non-linearities are indeed present due to the coupling mechanism employed in the manual micromanipulator between the manual joystick and the mirror assembly. These non-linearities render equation (2) unusable and are discussed further later in this paper.

The project goal was to develop a medical robotic system that will address some of the more onerous ergonomic problems associated with the mechanical micromanipulator[7] and lead to some level of automation. The issue of scaleability, the corresponding need for small, precise control inputs, and the issue of user comfort were the primary design areas of focus.

After considering a number of different human computer interfaces (HCI)[1] it was decided to employ a classic gaming type of electronic joystick (Saitek Cyborg Evo Force[5]) as the primary interactive user interface. Initially the design concept was to use the electronic joystick to allow the user to provide system inputs that would be translated

[5] http://www.saitekusa.com/prod/evoforce.htm

into surgical laser aiming. These joystick inputs were then manipulated under software control and forwarded to a slave controller that actually moved the micromanipulator. A variety of electromechanical devices were evaluated for the task of actuating the micromanipulator. Stepper motors, piezo-electric actuators, and DC servomotors were all investigated. HITEC HS-81[6] DC servomotors were ultimately selected for reasons of cost, accessibility, and simplicity of implementation. In addition, DC servomotors generally allow for position, velocity, acceleration, and force control. These self-contained devices require only power, ground, and a signal input. The full architecture of the medical robotic system consists of a hardware/software controller coupled with a mechanical assembly that mounts to the micromanipulator. A basic block diagram of the system is shown in Fig. 10.

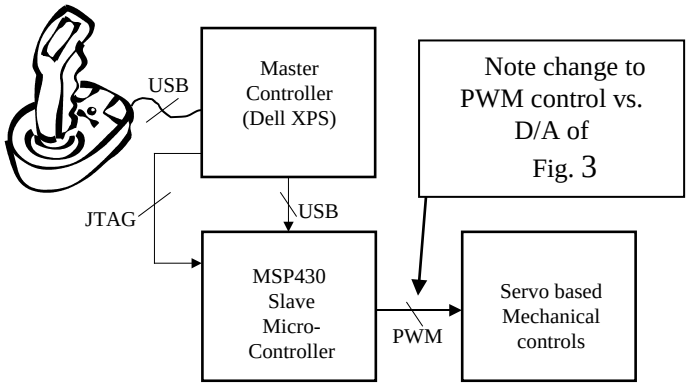


Fig. 10. Medical Robotic System Block Diagram

[6] http://www.hitecrcd.com/servos/show?name=HS-81

The user interface is implemented in MATLAB™ version 7.0 as a Graphical User Interface (GUI) running on the master controller laptop computer. The GUI provides the user with the ability to tailor the system operation to his/her preferences with respect to certain operational parameters. The master control program also interprets the joystick position and feeds this information in the form of x-y coordinates over a dedicated USB port to the slave micro-controller. Similarly to the first system design the slave micro-controller is based upon an Texas Instruments MSP430 processor reference design[8] and provides a USB serial communications link to the master controller. A JTAG port is used to download new firmware to the micro-controller and also for debug purposes. The slave micro-controller drives two PWM signal lines, one for x position and one for y position. These PWM lines in turn direct the operation of two servomotors that control the mechanical assembly attached to the micromanipulator.

The essential function of the slave microcontroller is translation of the coordinates received from the master controller into servomotor commands. The need to create PWM control signals for the servomotors lead to an evaluation of a number of different standalone microcontrollers available. Once again the MSP430 processor family was chosen for its versatility and because of the rich development environment available for these devices[9]. The next consideration was implementation of the messaging interface between the master controller and the slave microcontroller. The ubiquity of the Universal Serial Bus (USB) standard made this protocol the obvious choice for the

interface. The master controller had a number of available USB ports by virtue of the fact that it was based on a laptop PC. Having decided to employ USB for the messaging the next decision was how best to implement this functionality on the slave microcontroller. After reviewing various discrete design approaches for USB implementation[10], a reference design[8] was chosen as a base platform for several reasons. First, it provided a proven, stable starting point for an integrated USB interface design employing an MSP430 processor. It also enabled the use of off the shelf development tools such as the IAR Microsystems Embedded Workbench™. Lastly, the design also lent itself to repackaging in a footprint that met the requirements of this application. The basic system architecture is depicted in Fig. 11.

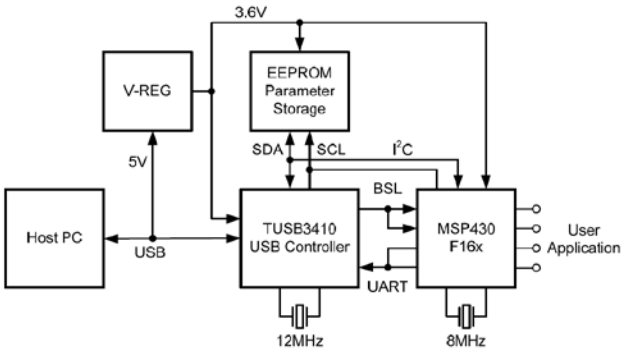


Fig. 11. Slave microcontroller block diagram[8]

Additional circuitry was designed and implemented to provide a separate high current +5V DC power supply for the servomotors. This circuit also powers a dedicated

linear voltage regulator to source the required +3.6V DC for the MSP430F169 and its related circuitry. As previously mentioned the slave microcontroller performs a translation function. It converts the x and y coordinate information received from the master controller (host PC) into appropriate PWM signals to alter the motion control servo positions. The control input to the servo is a pulse width modulated (PWM) logic signal. The servo is centered when it receives a pulse width of 1.5ms and is moved to its extremes of travel with 1ms and 2ms pulses respectively. Between these endpoints it is possible to continuously vary the servo position by means of the pulse width. The pulse train to the servo is refreshed approximately 50 times per second. The PWM signal lines were implemented using two of the I/O lines and one of the embedded timers on the MSP430. The schematic capture and circuit board layout were done using the EAGLE™ software package version 4.16 from CadSoft Online. The circuit schematic and the circuit board layout may be found in Appendix I and II. The circuit board layout was done manually using 0.007 inch track widths for the most part with thicker trace widths employed for the power circuitry. A two sided circuit board implementation was employed to minimize fabrication cost.

The system control software is distributed between the master controller and the slave micro-controller. The master controller software is implemented in a GUI running in MATLAB™ version 7.0. The slave micro-controller software is written in C and was

developed using the IAR Embedded Workbench™ IDE version 4.6.0.0 with version 3.40B of the C/C++ compiler.

The master control program was developed using the GUIDE functionality of MATLAB™. This is a simple and relatively easy to use mechanism for creating a GUI. The purpose of the GUI is to provide an ergonomic user interface for the setup and control of the robotic system. The GUI is by nature a collection of callback routines. Once a slider or toggle button is actuated the corresponding callback routine is executed and the GUI returns to its quiescent state. This simplicity obviates the need for a dedicated flowchart. A flowchart for the main joystick update timer routine described later is provided. The complete source code listing may be found in Appendix VII. The primary GUI screen is shown in Fig. 12.

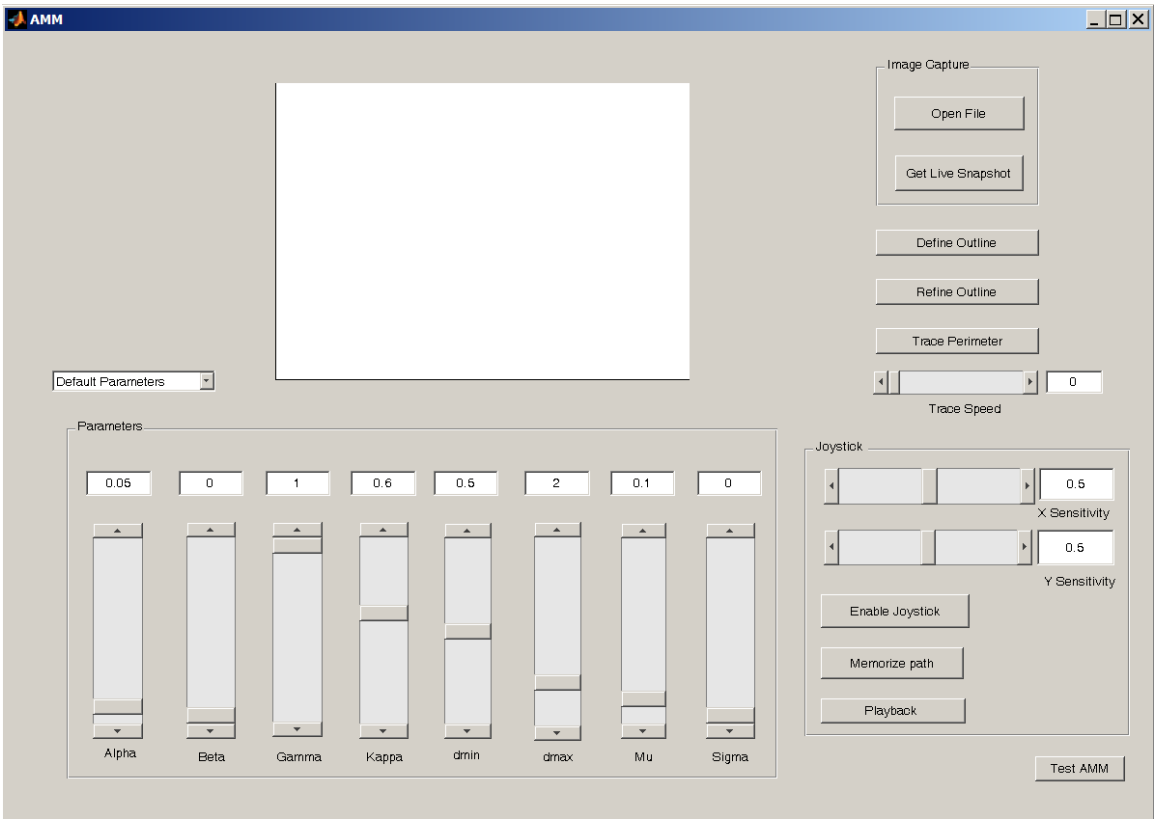


Fig. 12. Sample GUI control screen for robotic system

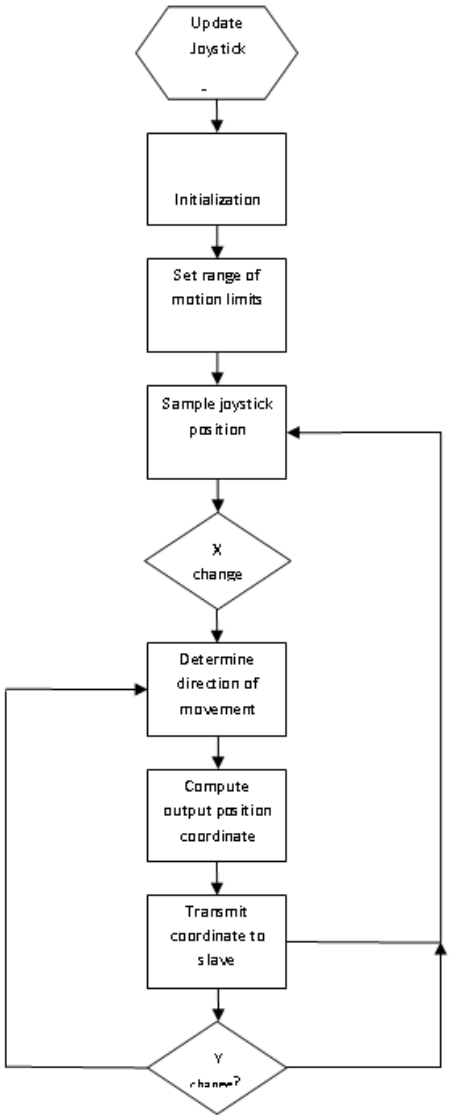


Fig. 13. Master Joystick Control Software Routine

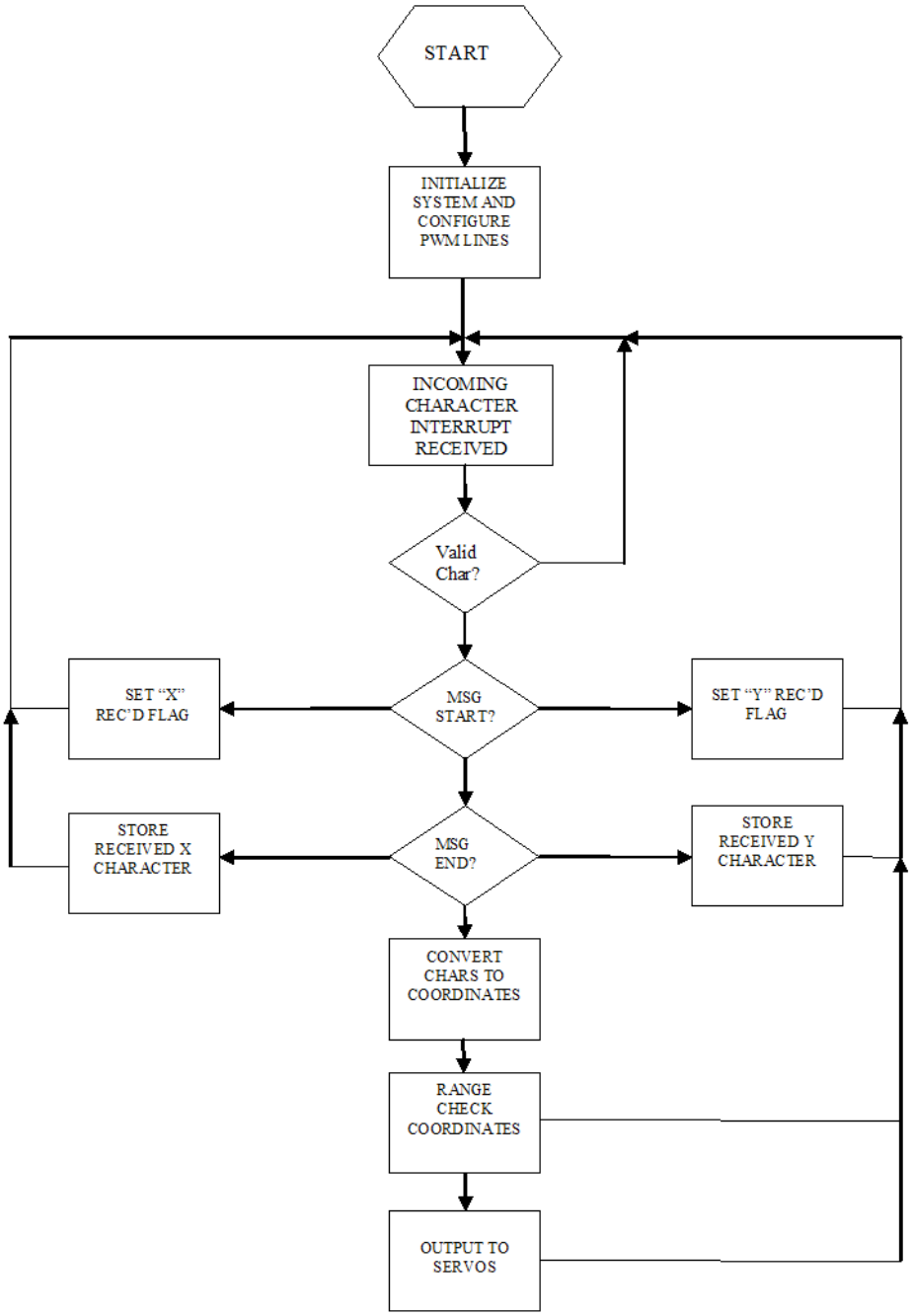


Fig. 14. Slave microcontroller Software Routine

The basic joystick user interface controls are located in a cluster in the lower right hand corner of the screen. By consolidating the adjustment functions into a relatively few slider and toggle button actuated controls, the interface is rapidly learned and easy to use. From this screen the user can enable/disable the joystick and set travel limits. The latter are useful when determining exclusion zones or for adjusting movement sensitivity. The user can also enable the path memorization and playback functions from this menu. The playback speed is adjustable via a slider control immediately above the joystick control cluster. The remainder of the screen is dedicated to the user interface and associated controls earmarked for future development activities.

The main software function of the master control program is the joystick service routine (see Fig. 13). The joystick position is determined using a timer routine that samples the joystick position every 80ms. This position information is then converted into x and y coordinates which are transmitted to the appropriate USB port to be sent to the slave micro-controller. If the path memorization feature is enabled in the GUI then the timer routine will also memorize each coordinate pair when the joystick trigger is depressed. These coordinate pairs may then be automatically repeated under user control at a later time.

The slave micro-controller runs an autonomous control program. This program consists of certain initialization and housekeeping functions combined with an interrupt service routine that receives and interprets characters. After initialization the

microcontroller sleeps until a character is received and an interrupt is generated as a result. Upon receipt of this interrupt the control program determines if the incoming characters are valid x and y coordinate messages. The valid characters are “X”, “Y”, the digits 0-9, and the carriage return (CR) special character. A valid message is composed of either “X” or “Y” followed by four digits and the CR as an end of message indicator. After a valid message is received the program performs an error check on the message to ensure that the received coordinate is within the valid range of travel for the servo. If this error check is successfully passed then the program translates the coordinate message into an appropriate corresponding PWM signal. This signal is then sent to the appropriate servo and the x or y mechanical actuator is moved. If an invalid character is received or a received message does not pass the error check then the offending character or message is discarded and the program returns to its initial state. The control program updates the x and y coordinates sequentially. A complete source code listing may be found in Appendix VIII.

The mechanical portion of the system was designed using SOLIDWORKS™ 2005 three dimensional mechanical modeling software. The most challenging portion of the mechanical assembly was the fabrication of the gimbal components. Fabrication of the gimbals themselves required the design and development of special precision machining fixtures. These fixtures were likewise developed and simulated using SOLIDWORKS™ and then produced using traditional machining techniques. Fig. 15

and Fig. 16 provide details of the completed fixtures. The raw gimbal stock is first formed to initial shape using the fixture of Fig. 15. Final forming is accomplished using a three ton arbor press in concert with the fixture of Fig. 16. Full details of the fabrication process may be found in Appendix IX.

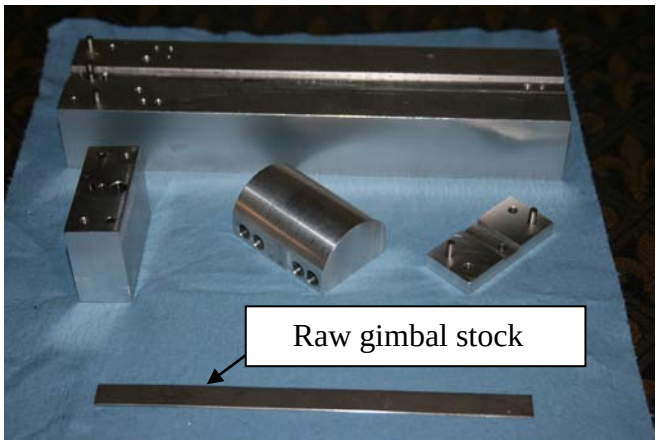


Fig. 15. Initial gimbal fabrication fixture components

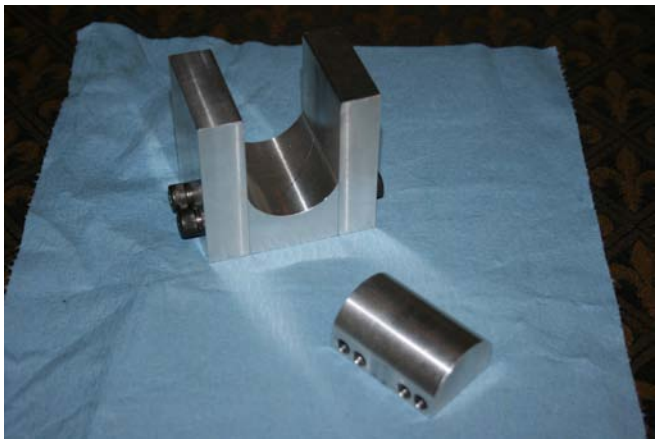


Fig. 16. Final gimbal forming fixture

The entire mechanical assembly was fabricated from 6061T6 aluminum with the exception of the gimbals and the mounting platform components. These components were made from 2024T3 aluminum in order to enable implementation of the necessary bends. Two gimbals were manufactured according to the preceding process in addition to the mounting plates and servo mounts. The mechanical drawings for these components and the manufacturing fixtures are found in Appendix V.

There were many fabrication challenges encountered in the course of the prototype development. The primary challenge throughout the design was the necessity of maintaining tight machining tolerances in order to minimize slop in the mechanism. The completed assembly of the initial mechanical prototype is depicted in Fig. 17.

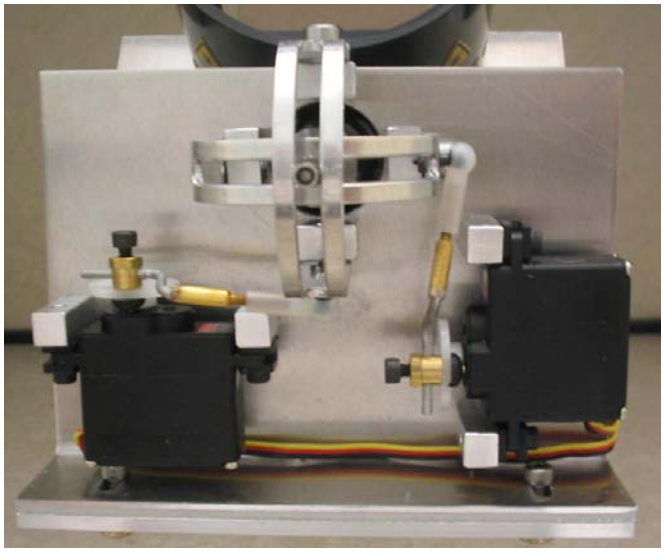


Fig. 17. Initial Prototype Medical Robotic System Mechanics

IV. INITIAL TESTING AND COMMISSIONING

The first step in system integration was building the slave micro-controller circuit board. After completing the circuit board, shown in Fig. 18 and Fig. 19, initial debug commenced.

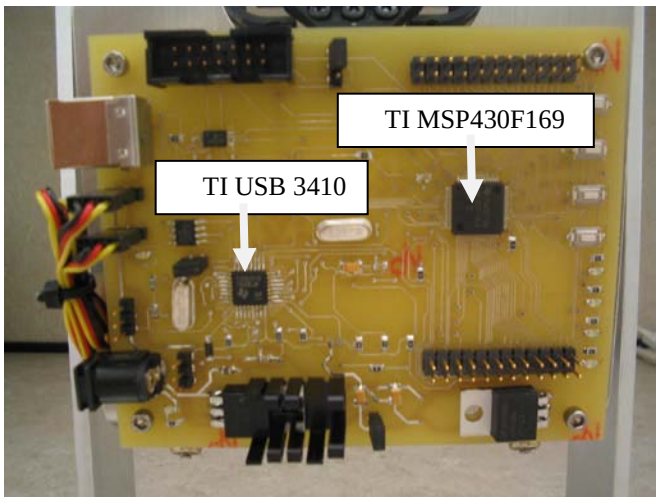


Fig. 18. Slave micro-controller circuit board

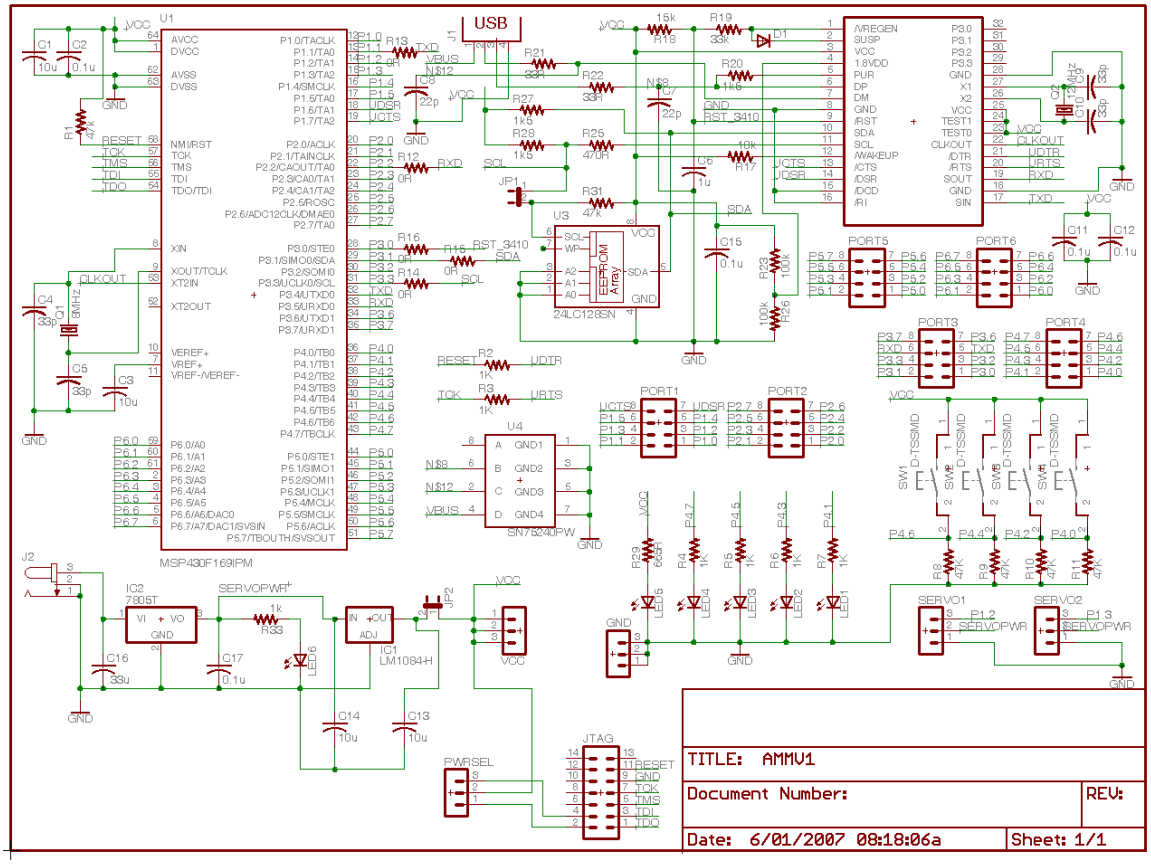


Fig. 19. Slave microcontroller circuit schematic

The first problem area was the initial power supply design. A complex switching power supply arrangement using DC-to-DC converters would not source enough current and therefore did not work. The final design employs two linear voltage regulators in cascade. The initial regulator converts the +15V DC input to the board into +5V DC to power the servomotors. This +5V DC signal is then also used to power a second linear voltage regulator that produces the +3.6V DC source required by the remainder of the

control electronics. This revised design has functioned well in initial prototype testing. Given the high continuous power consumption associated with certain automated tasks it will be necessary to revise this circuitry for future iterations of the system.

The next design hurdle was determining the appropriate settings for the MSP430 on board timers and related registers to implement the PWM signals required for the servo control. It is critical that the generated pulse widths remain in the range between 1 ms and 2 ms or the servomotors will behave erratically. The PWM software functionality was designed, implemented, and then cross checked using an oscilloscope with digital measurement capability to verify that the generated pulse widths were within the allowable range.

The slave micro-controller algorithm was developed and tested next. It was implemented as described previously; i.e. it is an interrupt service routine that examines the incoming character stream, decodes and validates messages, and then sends appropriate PWM commands to the servomotor(s) based upon these messages. The initial versions of this program were tested by using Hyperterminal™ to send manually formatted coordinate messages from a host pc to the micro-controller. This was useful as it decoupled the development and testing of the master controller software from the slave microcontroller efforts.

The mechanics also created numerous challenges and opportunities for improvement. The initial gimbal design incorporated simple flat machined slots as a bearing surface for the joystick cap (Fig. 24). While simple this also turned out to be

dysfunctional due to the friction induced by the geometry of the universal joint. To solve this issue it was necessary to machine crowns on the interior surfaces of the gimbal slots in order to achieve appropriately smooth movements. The crowns establish a single point bearing surface for the joystick cap and thus greatly reduce operational friction and binding.

The servo motor output wheel moves in response to its PWM control signal. The wheel is centered when a pulse of 1.5 ms is received by the servo motor and it reaches its limits of movement for pulses of 1.0 ms and 2.0 ms, respectively. Within the 1 ms control pulse range the output is almost continuously variable based on pulse width. This provides the ability to make very minute adjustments in laser aiming position and is a crucial feature of the medical robotic system. The angular displacement of the servo output arm corresponds to the angular displacement of the x and y mechanical gimbals of the system since they are connected by means of rigid pushrods as seen in Fig. 22. The angular displacement of the servomotor output arm is directly related to the input control pulse width as described above. Thus it is straightforward to predict the theoretical gimbal displacement and corresponding change in laser aiming point by means of the servo control signal pulse width.

V. EXPERIMENTAL RESULTS

The goal of the first experiment is empirical verification of the movement model for the medical robotic system. More specifically the goal of the experiment is to

compare the predicted change in laser aiming point with the actual change that is measured. For purposes of this experiment several cardinal points of operation were chosen to verify the model accuracy. The system range of motion was divided into quarters and each axis was evaluated at four points in each quadrant of operation. Predicted aiming points were then calculated using Equation (1) of the aiming point model. The results are summarized in the Predicted Displacement column of Table 2. The mechanical limits and range of motion for each axis are also found in the Measured Displacement column of Table 2. The system demonstrates the ability to traverse greater than +/- 0.80 inches in the y-axis which is adequate given the typical 21mm maximum vocal fold length. The x-axis is mechanically limited to less total throw by virtue of the x-axis gimbal location inside that of the y-axis gimbal. Despite this limitation the x-axis throw of 0.72 inches left and 0.53 inches right is more than adequate for our needs in laser phonomicrosurgery. The asymmetry of movement for both the x and y axes is discussed later in this paper. Additional measurements were performed to assess return-to-center performance and it was observed that the mechanics would reliably re-center when a system self-test was performed. This self test moves the mechanics to each extreme position and then returns to center. It was observed that simple excursions from center to outlying points and then returning to center did not re-center the system consistently.

Next, the medical robotic system was configured on a laboratory workbench in a standalone fixture. This fixture enabled observation and measurement of the movement

of each servomotor and its associated gimbal. A dual channel oscilloscope was attached to the system to monitor the servomotor control signal pulse widths for each gimbal. The oscilloscope provided an accurate pulse width measurement tool that was used to determine the servo motor angular displacements. This pulse width measurement tool was used to determine when the system was at each of the positions defined in the Predicted Displacement column of Table 2.

A surgical laser was then attached to the system and a target area established at the typical 400 mm operative distance employed in laser phonomicrosurgery. ZAP-IT® Laser Alignment Paper[7] was positioned in the target area in order to record changes in the laser aiming point. See Fig. 20.

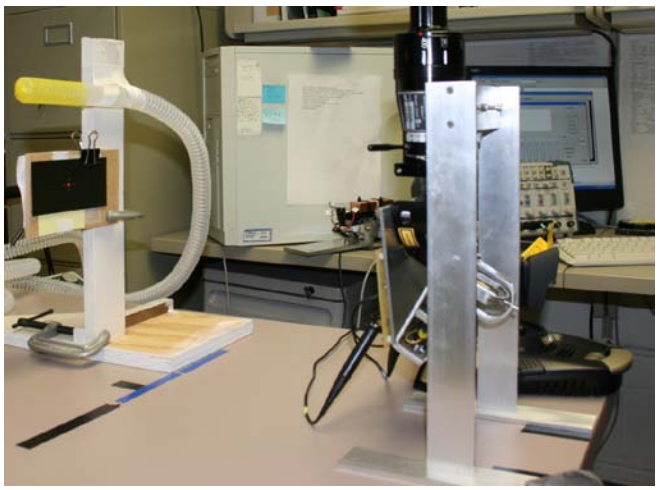


Fig. 20. Model Verification test setup

[7] ©2002 Zap-IT Corporation. All rights reserved. ZAP-IT® is a registered trademark of the Zap-IT Corporation.

The medical robotic system was then manually sequenced using a special software test program through the series of Predicted Displacement laser aiming points. Moving in quarter step (total throw in a given direction divided into four equal steps) increments, the laser was fired at each point in order to mark the laser alignment paper. The results are shown in Fig. 21 and Table 2.

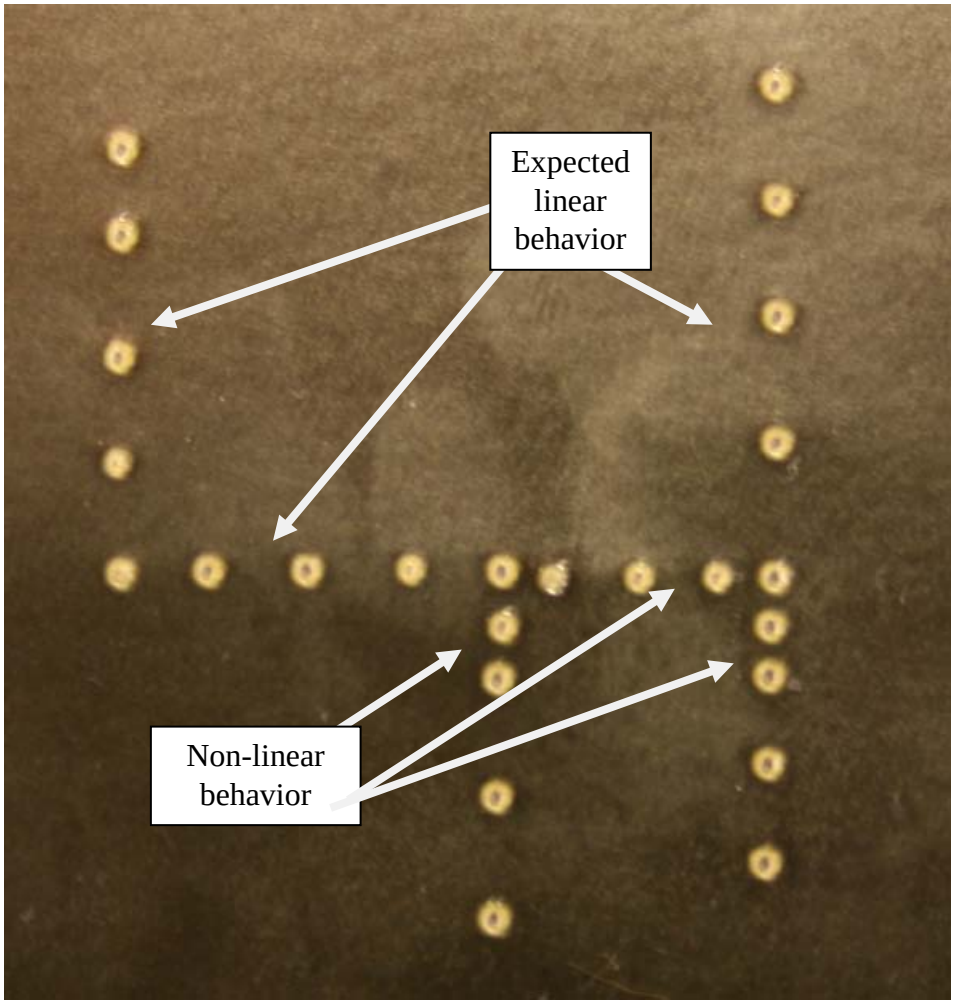


Fig. 21. Model verification test point pattern

At the conclusion of the firing sequence, measurements were made from the laser alignment paper of the changes in laser aiming points. These measurements were then collated and placed in the MRS Performance column of Table 2. A percentage error calculation was made for each measured displacement using the associated predicted

displacement as the reference data. The results are summarized in Table 2 and will be discussed later in this paper.

Table 2. Medical Robotic System Displacement Model Verification

	Joystick angular displacement	Theoretical aiming beam displacement	MRS Performance	Joystick scaling factor	Theoretical Mirror angular displacement	Theoretical predicted aiming beam displacement $x = 2r\sin\left(\frac{\theta}{2}\right)$	Error
one quarter left	5.250	0.208	0.188	0.90	0.682	0.188	0.01%
half left	10.500	0.415	0.375	0.90	1.364	0.375	0.00%
three quarter left	15.750	0.623	0.563	0.90	2.047	0.562	-0.01%
max left	21.000	0.830	0.750	0.90	2.729	0.750	0.00%
center	5.250	0.000	0.000	0.000	0.000	0.000	-
one quarter right	5.250	0.208	0.094	0.45	0.682	0.188	50.00%
half right	10.500	0.415	0.281	0.68	1.364	0.375	25.00%
three quarter right	15.750	0.623	0.422	0.68	2.047	0.563	25.00%
max right	21.000	0.830	0.531	0.64	2.729	0.750	29.17%
center	0.000	0.000	0.000	0.000	0.000	0.000	-
one quarter up	5.250	0.385	0.219	0.57	0.801	0.220	0.60%
one half up	10.500	0.770	0.438	0.57	1.602	0.440	0.60%
three quarter up	15.750	1.155	0.672	0.58	2.402	0.660	-1.77%
full up	21.000	1.540	0.844	0.55	3.203	0.880	4.15%
center	0.000	0.000	0.000	0.000	0.000	0.000	-
one quarter down	5.250	0.385	0.125	0.32	0.801	0.220	43.20%
half down	10.500	0.770	0.219	0.28	1.602	0.440	50.30%
three quarter down	15.750	1.155	0.938	0.81	2.402	0.660	-42.01%
full down	21.000	1.540	0.688	0.45	3.203	0.88	21.90%

VI. CONCLUSIONS AND FUTURE WORK

System fidelity was the first performance characteristic that was evaluated. In this context fidelity is defined as the ability of the system to faithfully implement the command path described by the user. Experiments with the original pushrod design revealed that the system performed well in manual mode (aiming guided using joystick) but less well in automated mode (system "plays back" points memorized during manual mode). The basic reason is that the user closes the feedback loop in manual operations. In any given movement, if the laser aiming point is not yet where the user desires then he/she simply continues providing appropriate control corrections using the joystick until the desired laser aiming point is achieved. Unfortunately no such feedback loop exists at this time for the automated operational mode. The automated playback function was initially less useful than hoped. This was due in large part to the observed inability of the controller mechanics to reliably re-center to a given starting point. This caused significant deviations between the user's programmed path and the path observed during automated playback, primarily in the y-axis direction. The original pushrod design was chosen for its simplicity and the readily available off-the-shelf components. The ball and socket implementation was intended to provide a low friction connection to the gimbals that would also allow for some minor misalignment between the plane of the servomotor output arm and the plane of the associated gimbal. Some of the possibilities for

resolution include: (1) redesign of the mechanics enabling direct servo drive in lieu of the current pushrod configuration, or, perhaps (2) the addition of optical shaft encoders to each gimbal. The latter approach would also require additional software development but would theoretically enable a high degree of fidelity in repetitious tasks, such as playback of a memorized excision path. Given the significant required resource investments for each of the foregoing approaches, a short term initial corrective action was taken in the form of a new pushrod design. The new pushrod design greatly reduced the play present in the initial ball and socket approach and improved general movement precision and fidelity. This was accomplished through the use of precision shoulder screws at the servomotors and gimbal pivot points combined with tighter machining tolerances (+/- 0.001 inches). As with the gimbal mounts the shoulder screws employ Delrin™ washers to facilitate movement smoothness and prevent metal to metal contact between the components. The revised mechanical design is shown in Fig. 22.

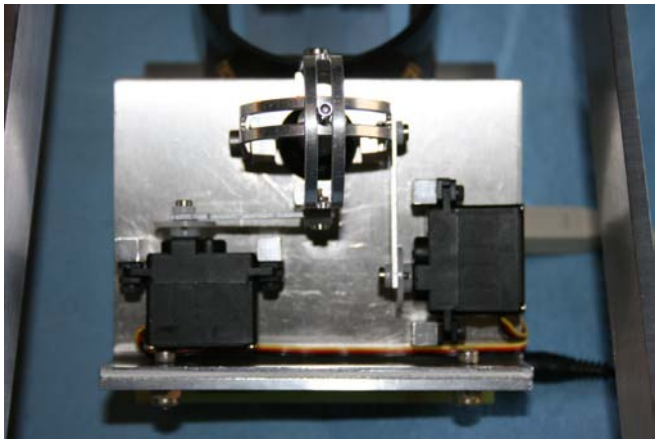


Fig. 22. Final Prototype Mechanical System

Details of the new pushrod design are found in Appendix V and a detailed photograph is shown in Fig. 23.

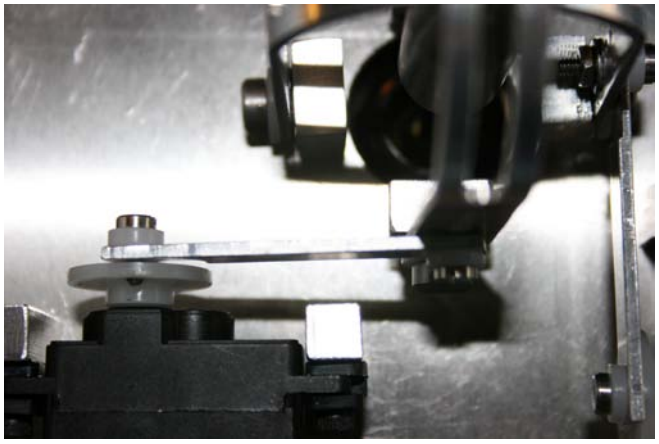


Fig. 23. X-axis pushrod detail view

The full scale travel for the x-axis is 1.281 inches and 1.531 inches for the y-axis. Inspection of the measured displacement data of Table 2 and the recorded performance data of Fig. 21 reveals some small asymmetries in the left side x-axis travel data. These differences are most probably due to mechanical tolerances. Unfortunately the right side x-axis travel shows significant compression of the range of movement as well as the accuracy relative to the commanded control input. Similarly in the y-axis the upward movement quadrant exhibits good performance relative to the model while the downward

quadrant has significant compression of the range of movement. Examination of the geometry of the new pushrod design reveals the probable cause of the asymmetries. The pushrods should be orthogonal to their respective gimbals in the neutral position to ensure symmetric travel ranges in each direction. Examination of Fig. 23 shows an upward tilt to the pushrod affixed to the x-axis gimbal. There is a similar geometric anomaly for the y-axis pushrod. Measurement technique and the potential associated variance were also contributing factors but of lesser significance than the geometric issue. Clinical experiments conducted later in this work did not reveal any significant performance impairment perceived by the clinicians conducting the evaluations. The presence of the human operator compensated for the error induced by the geometric anomaly. This self correcting feature would of course be absent in the automated operations contemplated in chapter 4 of this work. In that case the asymmetries introduced by the pushrod offset would be a significant performance issue in certain cases. The worst case would be for extensive outlines that encompassed areas well away from the neutral aiming position. For these situations a remedy must be implemented. The most straightforward solution is of course the alteration of the pushrod geometry to create an orthogonal relationship between the pushrod and the gimbal in the neutral position. However, there are more sophisticated solutions to the anomaly. An example of one such solution would be the addition of optical shaft encoders to each of the gimbals. This would enable precise position control as a result of the feedback that would be provided.

Next, a subjective evaluation and physical inspection of the mechanics also reveals some differences in the mechanical tolerances between the joystick cap and the x and y gimbal interfaces. Refer to Fig. 24 for details.

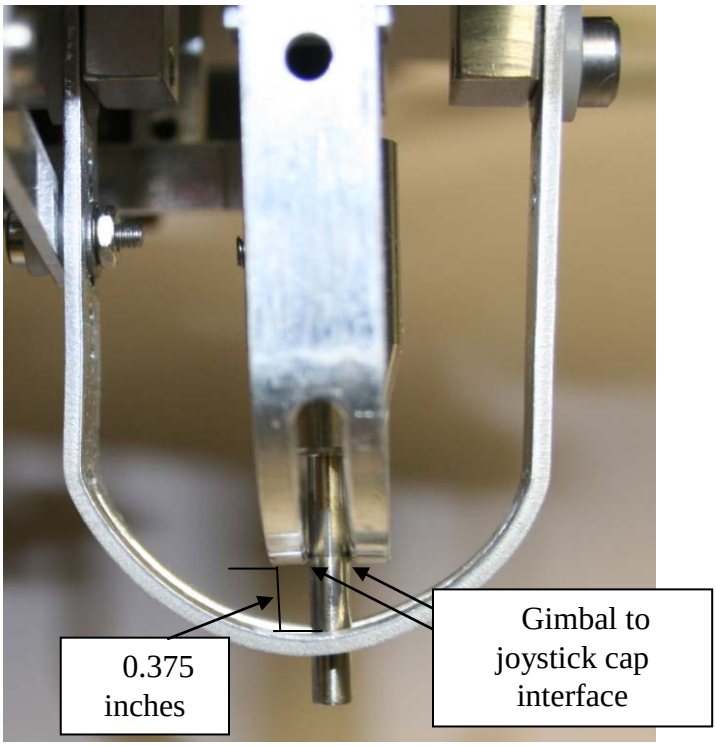


Fig. 24. Gimbal interface detail

The measured displacements of Table 2 were made manually using a machinist's scale. In each case the displacement was measured from the center point of the appropriate axis to the center of the laser aiming point (see Fig. 21). The predicted displacements of Table 2 were calculated in accordance with the mathematical model

described previously in this paper. As previously mentioned it was necessary to discard the initial formulation of μ, the scaling factor relating manual joystick displacement to changes in laser aiming point. Due to the complex mechanical arrangement of the mechanical micromanipulator, including two different internal pivot points, the relationship between angular displacement of the joystick and angular displacement of the mirror assembly is not linear. Discussions with the manufacturer of the micromanipulator confirmed this non-linear relationship. Unfortunately the manufacturer did not have a defined relationship for this parameter but rather depended upon subjective physical evaluations of the completed units. More particularly, for a given operative distance (in this case 400mm) the beam path was evaluated and nominal measurements of the maximum x-axis (1.66 inches) and y-axis (3.08 inches) excursions were noted. These maximum excursions were then scaled for the travel available in the MRS. This new scaling factor was then used as the basis for predicting changes in the mirror assembly angular displacement. The mirror assembly angular displacement was then used in equation (1) to predict aiming beam displacement changes. This is the information summarized in Table 2. We observe reasonable fidelity in the left x-axis and upward y-axis. The asymmetries in the right x-axis and downward y-axis are largely due to the pushrod geometry issue previously described. Other potential causes of error in the aiming model relate to the inherent gimbal geometry (see Fig. 24). Given the required radii of the gimbals it is not possible to arrange them as in Fig. 8. The separation between them is necessary to assure mechanical clearance while allowing an acceptable

range of motion. One consequence of this situation is that the x-axis gimbal is located significantly further away (approximately 0.375 inches) from the center of rotation of the mechanical joystick than is the y-axis gimbal. This additional distance is believed to introduce error into the basic assumption of the model, i.e. that the end of the mechanical joystick is moving in a linear fashion in a plane rather than as a point on a hemispherical surface. The further away the point of interest lies from the center of rotation, the less planar the movement of said point and the more significant the error. Further refinement of the model to more accurately describe the mechanical motion is the most straightforward corrective action that might be undertaken in future work.

Having proven the basic operation of the medical robotic system design and its associated mathematical model, the next steps are further operational tests. These tests will be conducted in a clinical environment and will assess ease of use, accuracy, reliability and the overall potential utility of the system as an alternative to the classic mechanical micromanipulator.

REFERENCES

[1] J. F. Giallo, II; Edward Grant; Robert Buckmire, "An Evaluation of User Interfaces for Laser Phonomicrosurgery," 2008. Submitted to Transactions on Information Technology in BioMedicine. Reference number, TITB-00173-2008.

[2] R. Buckmire, M. D., 2006, p. Image provided courtesy of Dr. Buckmire.

[3] E. Aoki, E. Aoki, T. Suzuki, E. Kobayashi, I. A. S. I. Sakuma, K. A. K. K. Kozo, and M. A. H. M. Hashizume, "Modular Design of Master-Slave Surgical Robotic System with Reliable Real-Time Control Performance," in *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*, 2006, pp. 80-86.

[4] V. F. Munoz, V. F. Munoz, I. Garcia-Morales, C. Perez del Pulgar, J. M. A. G.-D. J. M. Gomez-DeGabriel, J. A. F.-L. J. Fernandez-Lozano, A. A. G.-C. A. Garcia-Cerezo, C. A. V.-T. C. Vara-Thorbeck, and R. A. T. R. Toscano, "Control movement scheme based on manipulability concept for a surgical robotic assistant," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, 2006, pp. 245-250.

[5] R. H. Taylor and D. Stoianovici, "Medical robotics in computer-integrated surgery," *Robotics and Automation, IEEE Transactions on*, vol. 19, pp. 765-781, 2003.

[6] R. H. Cannon, *Dynamics of physical systems*. New York,: McGraw-Hill, 1967.

[7] R. H. Ossoff, J. A. Coleman, M. S. Courey, J. A. Duncavage, J. A. Werkhaven, and L. Reinisch, "Clinical-Applications of Lasers in Otolaryngology - Head and Neck-Surgery," *Lasers in Surgery and Medicine*, vol. 15, pp. 217-248, 1994.

[8] A. Dannenberg, "MSP430 USB Connectivity Using TUSB3410," in *Texas Instruments Application Report*. vol. SLAA276A, 2006.

[9] T. Instruments, "MSP430 Development tools." vol. 2008, 2008, p. http://focus.ti.com/mcu/docs/mcuprodtoolsw.tsp?sectionId=95&tabId=1203&familyId=342&toolTypeId=1.

[10] J. Hyde, *USB design by example : a practical guide to building I/O devices*, 2nd ed. [Hillsboro, Or.?]: Intel Press, 2001.

CHAPTER 3. The Testing and Evaluation of a Medical Robotic System for Laser Phonomicrosurgery

Abstract—The prevalent traditional methodology within phonomicrosurgery for remote control of the surgical laser is the mechanical micromanipulator. This paper overviews the design of a medical robotic system specifically designed for automating parts of laser phonomicrosurgery procedures. Performance metrics were obtained from clinical evaluation trials with surgeons that possessed a range of experience. User evaluations demonstrated that neophyte clinicians preferred the new medical robotic system over the traditional micromanipulator and they also performed simple tasks with greater accuracy using the new system.

I. INTRODUCTION

Previously the state of the art for surgical laser user interfaces in phonomicrosurgery was examined in a paper that reviewed the area, see Giallo *et al.* [1] It was found by examining the field that there was an opportunity to improve upon the design and implementation of the traditional user interface employed to remotely aim a surgical laser. The traditional user interface is the remote mechanical micromanipulator[8] depicted in Fig. 1. The challenges associated with employing the mechanical micromanipulator began with a study of its ergonomics. The typical operative setup for phonomicrosurgery is shown in Fig. 2. The surgeon is seen using special armrests to steady their hands and wrists while employing traditional "cold steel" surgical instruments. Unfortunately, these supports are not practical aids when using the mechanical micromanipulator. The mechanical micromanipulator is affixed to the base of the surgical microscope optics and the manual joystick seen on the left side of Fig. 2 then extends outward in space beneath the microscope. The surgeon is required to move the end of the manual joystick to alter the surgical laser aiming position. There is no physical support available to steady the surgeon's arms or hands.

[8] UniMax Microspot Micromanipulator, PhotoMedex, Inc., Montgomeryville, PA 18936

Fig. 1. Mechanical micromanipulator[2]

In addition the manual joystick itself has limited adjustments for user preferences. It is possible to adjust the resistance to motion via a tension adjustment (the ring around the base of the manual joystick) but this is the only user adjustable parameter. Additionally the operative field for the laser is quite small, with the size of the vocal fold on the order of 11-15mm for females and 17-21mm for males [3]. As a result very small, precise changes in the joystick position are necessary to accurately alter the surgical laser aiming point. These movements are very difficult to make consistently and are a particular challenge to novice clinicians. Circumstances become even more problematic when ablating significant tissue masses. In this case the clinician is required to make multiple passes over the same target areas. As ablation progresses the visual landmarks change and in some cases are destroyed. The challenge is to accurately ablate the intended tissue without harming surrounding healthy areas [4-6]. All of these issues lead

the authors to conclude that some form of robotic assistance will prove beneficial to the clinician practicing laser phonomicrosurgery. A medical robotic system was designed and fabricated for this purpose. This system has been tested and evaluated to determine its engineering performance metrics and reliability, see Giallo *et al*. [7]. Here it is tested and evaluated by clinicians to determine performance metrics and to obtain end user feedback related to its design.

This paper describes the development and application of this system during clinical evaluations to assess its ease of use, accuracy, reliability, and efficiency.

Fig. 2. Phonomicrosurgery operative environment[2]

II. EXPERIMENTAL RESULTS

Initial experiments involving the medical robotic system were conducted to assess basic usability and operational characteristics. First, nine subjects were recruited from an available population of surgeons from the field of otolaryngology. The surgeons were of differing skill levels and clinical experience, ranging from little or no expertise with use of the classic micromanipulator up to experts in laser phonomicrosurgery. For this experiment the medical robotic system was set up in a free standing mode, i.e. not integrated with a surgical microscope. No visual magnification was provided. The system was, (1) placed in a mounting stand on a table with the user interface joystick positioned to the right of the system, and (2) connected to a CO_2 surgical laser via the micromanipulator laser port. The subjects were seated behind the appliance such that they could comfortably operate both the joystick and the laser firing foot switch while simultaneously visualizing the laser target area mounted on an adjacent wall. The target area was located at the 400 mm operative distance normally employed in laser phonomicrosurgery. Each subject was identically briefed on the operation of the medical robotic system. No formal training on the system was provided to the subjects in advance of the experiment since the goal was to evaluate accuracy and ease of use. However, each subject was allowed to experiment with the system to gain an operational "feel" for the appliance prior to beginning the actual assigned task.

The detailed instructions provided to the surgeons as well as the written questionnaire used to log the experimental data may be found in Appendix VI. This paper summarizes these detailed instructions as follows: Each surgeon was instructed to perform two basic dexterity tasks using the system. These tasks involved moving the laser aiming point in sequence among several predetermined points that were marked on a target. At each point the subject was instructed to mark the point by both manually firing the laser using a foot switch and then depressing the joystick trigger to electronically mark the point. These operations were performed sequentially and each target point was dealt with in this fashion. After all the points had been visited the subject was asked to complete a written survey evaluating their experience with the medical robotic system. The survey was loosely based upon the System Usability Scale (SUS)[8] and was designed to assess initial user impressions of the system. Usability is a subjective concept and has a number of aspects. In the SUS these aspects include effectiveness, efficiency, and satisfaction. Brooke[9] defines effectiveness as the ability of users to complete tasks using a given system as well as the quality of the completed work. Efficiency is defined as the level of resource that is consumed in completing a given task. Satisfaction is the most subjective metric and it attempts to measure whether the experience of using the system was satisfactory to the user. Varying interpretations of these aspects arguably make it very difficult to compare two systems let alone declare one “more useable” than another. Each individual user brings his/her life experience and

training to the evaluation of a new system. This background is an unavoidable, if intangible, element of the evaluation process that has a significant effect on the outcome. The survey form provided a free form comment section to allow participants to express their feedback on points that the questionnaire did not explicitly cover. These comments were very useful in formulating plans for the ongoing enhancement of the appliance. These comments are addressed in the discussion section of this paper. The survey results themselves are summarized in Table 1.

Table 1. Summary of initial usability survey results. A score of 5 indicates strong agreement that the system possesses the stated attribute while a score of 1 indicates strong disagreement.

	Joystick ease of use	Appropriate force required	Trigger ease of use	System ease of use	Inconsistency in operation	Cumbersome	Laser case experience
Subject 1	1	2	1	1	5	5	10-100
Subject 2	5	4	5	5	1	1	10-100
Subject 3	5	5	4	4	2	2	--
Subject 4	3	5	5	5	-	1	0-10
Subject 5	4	5	5	5	2	3	0-10
Subject 6	3	5	4	4	1	1	0-10
Subject 7	5	2	4	3	4	2	0-10
Subject 8	5	5	5	5	5	5	0-10
Subject 9	1	1	1	1	1	1	--
Average	3.6	3.8	3.8	3.7	2.6	2.3	

After the initial usability study was conducted, additional experiments were performed to assess basic control and ergonomic characteristics of the appliance. The objective of these experiments was to obtain quantifiable performance data from a group of subjects performing a common task. Prior to conducting the experiment it was necessary to design and construct appropriate targets for use in the evaluations. A number of different shapes and sequences were considered in this process. The first approach to the problem was to employ circular patterns. These patterns were drawn on ZAP-IT® Laser Alignment Paper[9] using opaque paint marker pens from a craft store. Several circles were drawn on each individual test card as seen in Fig. 3. Individual highlighted points were marked at various locations along the perimeter of the circular patterns. In this particular test card, two alternative testing techniques were evaluated as well as different laser power settings. The center circle pattern was used to assess the difficulty of targeting individual points on the perimeter of the circle. The two outside circles were used to evaluate the difficulty of following the circumference of the pattern while the surgical laser was fired continuously. The leftmost circle was done using a 2W power setting on the surgical laser while the rightmost circle was done with a 5W power setting. It was found that the 5W power setting produced the most visible laser track albeit at the cost of a wider pattern. Additionally, following the circular track while the laser was fired continuously was found to be more difficult than targeting individual

[9]

points. Targeting individual points was a trivial exercise regardless of their position on the target or sequence of targeting.

Fig. 3. Test pattern candidates

Additional patterns were also evaluated in this process. Some of these candidate patterns are shown in Fig. 4 and Fig. 5.

Fig. 4. Linear and Circular test pattern candidates

Fig. 5. Circular and serpentine test pattern candidates

The targets of Fig. 4 and Fig. 5 were prepared in similar fashion to those of Fig. 3 using the same materials and methods. In these cases it was desired to evaluate the effect of smaller targets as well as the relative difficulty posed by linear targets versus circular targets. It was found that circular targets are more challenging than linear targets and that the challenge posed is inversely proportional to the size of the target. Thus the smaller, circular targets were the most difficult for the test subjects to accurately gauge. Further investigation was done and a hybrid shape was developed for use in the actual experiment (see Fig. 8). The basic concept behind this shape was to provide a challenging pattern which contained varying degrees of difficulty. The broad, sweeping curve of the left side of the target was intended to be easier to navigate than the tighter area at the rightmost portion of the target. This shape was chosen because curved motions are commonly employed in certain ablation procedures.

Six subjects were recruited from an available population of surgeons. The subjects were of differing skill levels and clinical experience and all were naive users of the classic laser micromanipulator. Most of them did not participate in the initial

usability study documented in Table 1. As explained earlier for this experiment the medical robotic system (MRS) was set up in a free standing mode, i.e. not integrated with a surgical microscope. Similar to the last experiment the appliance was placed in a mounting stand on a table at the 400mm operative distance with the user interface joystick positioned on either side of the appliance according to subject preference. A CO_2 surgical laser was connected to the micromanipulator laser port of the appliance in conventional fashion. The subjects were seated behind the appliance and chose their physical position such that they could comfortably operate the joystick while simultaneously visualizing the laser target area mounted on an adjacent wall. Again each subject was identically briefed on the operation of the medical robotic system. As in the previous experiment no formal training on the system was provided to the subjects in advance of the experiment since the goal was to evaluate ease of use. However, each subject was allowed to experiment with the system to gain an operational "feel" for the appliance prior to beginning the actual assigned task. The experimental setup is depicted in Fig. 6.

Fig. 6. Experimental setup

Each subject was then instructed to perform a basic dexterity task using the medical robotic system. This task involved moving the laser aiming point between two serpentine lines spaced approximately 4mm apart. Fig. 7 depicts both targets on the test card while Fig. 8 gives a close-up view of the MRS target and laser path trace.

Fig. 7. Sample test results. MRS on left and MM on right.

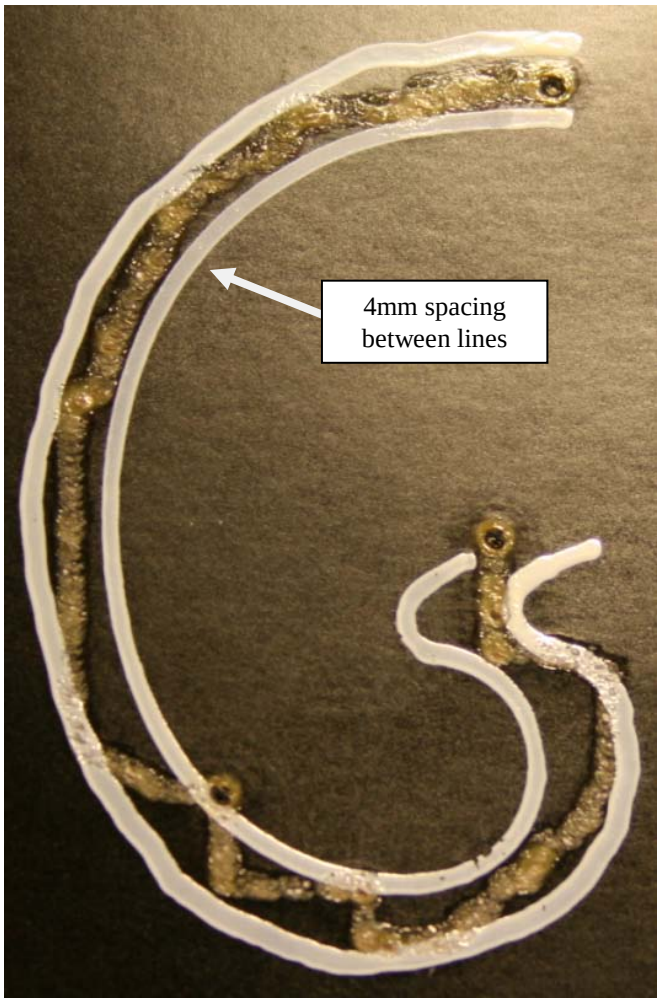


Fig. 8. Close-up view of MRS target and laser path trace

Firing of the laser was controlled by a laser safety technician and was commenced and terminated based upon oral commands from the subject. The subject instructed the technician to activate the laser when they were prepared to begin the test and then to deactivate the laser once they had completed traversing the test path. Subjects were

instructed to proceed at their own pace and the task durations were not timed. After all subjects had completed this task using the MRS, the test was then repeated using the classic mechanical micromanipulator (MM) in similar fashion. For ease of data tabulation both tests were run using one ZAP-IT® Laser Alignment Paper card per subject. Each card contained two identical targets, one for the MRS run and one for the MM run. The total path length was then measured and used as the frame of reference. Errors were recorded anytime the subject guided the laser outside of the target track. The length of each deviation was measured and the errors were aggregated and recorded for each subject. The process was repeated for the MM run in the same way. Accuracy was computed as the difference between the path length and the error divided by the path length. Accuracy calculations for both the MRS and MM runs were performed for comparative purposes. The results are summarized in Table 2. A dependent, paired data t-test analysis was also performed on these first pass statistics and is discussed later in this paper.

Table 2. Summary of initial controllability tests. Medical Robotic System (MRS) vs. Manual Micromanipulator (MM). Measurements are in inches.

Subject	MRS error	MM error	MRS Accuracy	MM Accuracy	D_i	$(D_i - \bar{D})^2$
1	1.375	2.094	60.4%	39.6%	-0.719	0.237
2	1.250	0.875	64.0%	74.8%	0.375	0.369
3	1.500	1.781	56.8%	48.6%	-0.281	0.002
4	1.094	0.969	68.5%	72.1%	0.125	0.128
5	1.094	1.375	68.5%	60.4%	-0.281	0.002
6	1.063	1.844	69.4%	46.8%	-0.781	0.302
7	0.563	0.625	83.8%	82.0%	-0.062	0.029
Average	1.134	1.366	67.3%	60.6%	-0.232	-
Total	-	-			-1.625	1.068

III. DISCUSSION

Table 1 shows a broad spectrum of results. Two of the experienced clinicians gave polar opposite system evaluations while the novices expressed near universal enthusiasm for the system. This finding is encouraging as it is the next generation of clinicians that will employ and refine this technology. Two of the novices likewise failed to follow the survey directions carefully (subjects 8 & 9) and thus gave conflicting answers to some of the questions. In addition to the quantitative grading of system attributes, the surveys also provided the opportunity for free form user feedback. This feedback was very useful and revealed how user expectations shape perception of system

usability. For example, the classic micromanipulator moves the laser aiming point upward in the operative field of view when the joystick is moved downward toward the user. Early clinical feedback during system development caused the authors to mimic this behavior in the medical robotic system. However, this movement correlating a downward joystick movement with an upward laser aiming point reaction was counterintuitive for at least one subject. When questioned about his discomfort, the subject explained that his background with joystick HCIs (Human Computer Interface) involved computer flight simulation programs. These programs mimic actual flight controls wherein a downward movement in the joystick causes the nose of the aircraft to pitch down rather than up. Thus the subject was frustrated in his attempts to aim the laser and felt that the HCI should have followed the flight control convention rather than the classic micromanipulator approach. In this case the authors felt that it was better to facilitate the transition of experienced clinicians to the use of the medical robotic system rather than mimic the more familiar HCI from an unrelated application. It is hoped that new users will readily adapt to the HCI as it is defined since they have little or no direct experience in laser phonomicrosurgery and therefore few habits and behaviors to modify in order to acclimate to the appliance. Additional experiments conducted over time would be needed to test this hypothesis.

Some of the negative feedback may be rooted in the fact that most of us are resistant to change. Often when confronted with a new method for accomplishing a

familiar task we rebel at the prospect of learning a new technique with uncertain benefits. Additional exposure and practice with the appliance may moderate these issues and further experiments conducted over time would be needed to determine if this is indeed the case.

While the subjects completed the written survey the authors evaluated the playback function of the appliance. For this portion of the experiment the laser was placed in continuous firing mode and the aiming playback function was invoked. The playback function automatically retargeted the laser aiming point on the locations that the subject had marked using the trigger function of the joystick. After the laser concluded firing, the target areas were examined. Specifically the authors evaluated the manually chosen target points against those chosen by the appliance in playback mode. The goal was that both sets of target points should be coincident. It was found that in nearly all cases some variation occurred in one or more of the playback target points. As previously mentioned a portion of the variance is directly attributable to the original ball and socket pushrod design employed in the initial prototype. However, this does not account for the entire error since upgrading to the final pushrod design did not completely solve the problem. Further investigation into the root cause of the problem is underway.

The second experiment brought some interesting information to light. In examining the first pass statistics of Table 2 we see that two thirds of the participants

scored higher using the medical robotic system than they did with the classic mechanical micromanipulator. The overall average performance metrics were somewhat similar with the medical robotic system enjoying a 7% advantage over the classic manual micromanipulator. In examining the average error of the initial statistics of Table 2 we observe that the participants had a 17% lower error rate using the medical robotic system. This is a noteworthy finding since it appears to indicate that the medical robotic system contributes to improved accuracy in the given task.

However, these initial findings are tempered by the statistical analysis that was also performed. A dependent, paired data t-test was conducted on this first pass error data[9]. Details of this analysis are as follows:

$$S_D^2 = \frac{1}{n-1}\sum_{i=1}^{n}(D_i - \bar{D})^2 \qquad (1)$$

$$S_D = 0.422$$

Assumptions:

$$n = 7, \alpha = 0.05, \mu_{D0} = 0$$

We formulate the null and alternative hypotheses as follows:

$$H_0: \mu_{MRS} = \mu_{MM}$$

$$H_A: \mu_{MRS} \neq \mu_{MM}$$

From Table 2 we obtain the variable values for equations (2) and (3) and solve:

$$\hat{\sigma}_{\bar{D}} = \frac{S_D}{\sqrt{n}} = 0.159 \quad (2)$$

$$t_{calc} = \frac{\bar{D} - \mu_{D0}}{\hat{\sigma}_{\bar{D}}} = -1.455 \quad (3)$$

We reject H_0 if either of the following conditions is met:

$$t_{calc} < -t\left(n-1, \frac{\alpha}{2}\right)$$

$$t_{calc} > t\left(n-1, \frac{\alpha}{2}\right)$$

From the t-tables of [9] we obtain t(6,0.025) = 2.4469. Thus, comparing this test statistic with the results of (3) it was found that the null hypothesis (H_0: $\mu_{MRS}=\mu_{MM}$) could not be rejected. Stated differently, from this first pass data there is no difference between the mean error of the MRS versus the MM.

Further experiments will be needed to determine if the MRS delivers a statistically determined benefit in terms of operative error. An action that will potentially benefit the statistical analysis and lead to a significant finding is increasing the number of users. This increases the number of degrees of freedom in the analysis and lowers the value of the reference t statistic provided in the statistical table. Additionally, given that these results were based on a single trial run per participant, the authors believe that additional practice and experience will improve the user's performance. Those users that did not initially score better using the medical robotic system may improve with practice. It is possible that they might improve to the point where their proficiency with the medical robotic system exceeds their early performances with the manual micromanipulator. Providing that this improvement does indeed occur and that it can be transferred to the clinical setting it is reasonable to conclude that operative outcomes will be improved.

IV. CONCLUSIONS AND FUTURE WORK

The experiments conducted to date have proven that the medical robotic system has promise as a tool in laser phonomicrosurgery. Initial indications are that it will contribute to improved operative outcomes by increasing surgical accuracy. Next steps focus on the further automation of certain tasks within laser phonomicrosurgery. Specifically, the identification of excision perimeters and the ablation of defined perimeters will be investigated. Previous investigation has indicated that the use of active contours to define excision paths is a promising implementation approach. Efforts to develop and integrate this capability into the medical robotic system will be undertaken in the near future.

REFERENCES

[1] J. F. Giallo, II; Edward Grant; Robert Buckmire, "An Evaluation of User Interfaces for Laser Phonomicrosurgery," 2008. Submitted to Transactions on Information Technology in BioMedicine. Reference number, TITB-00173-2008.

[2] R. Buckmire, M. D., 2006, p. Image provided courtesy of Dr. Buckmire.

[3] D. M. Bless and J. H. Abbs, *Vocal fold physiology :contemporary research and clinical issues*. San Diego, Calif.: College-Hill Press, 1983.

[4] R. A. Buckmire, MD; Associate Professor of Otolaryngology/Head & Neck Surgery, University of North Carolina; Director, UNC Voice Center, "Lasers in Laryngology," Chapel Hill, 2006, p. 21.

[5] K. K. H. Chao, E. Cheung, W. B. Armstrong, and B. J. F. Wong, "The effect of optical design on micromanipulator spot size using CO2 laser irradiation," *Otolaryngology-Head and Neck Surgery*, vol. 126, pp. 593-597, Jun 2002.

[6] M. S. Strong and G. J. Jako, "Laser Surgery in Larynx - Early Clinical Experience with Continuous Co2-Laser," *Annals of Otology Rhinology and Laryngology,* vol. 81, pp. 791-798, 1972.

[7] J. F. Giallo, II; Edward Grant, "Designing of a Medical Robotic System for Laser Phonomicrosurgery," 2008. Submitted to Robotics and Autonomous Systems. Ms. Ref. No.: ROBOT-D-08-00208

[8] J. Brooke, "SUS: a "quick and dirty" usability scale," in *Usability Evaluation in Industry*, B. T. P. W. Jordan, B. A. Weerdmeester & A. L. McClelland, Ed. London: Taylor and Francis, 1996.

[9] P. V. Rao, *Statistical research methods in the life sciences*. Pacific Grove, CA: Duxbury Press, 1998.

CHAPTER 4. Towards the Automation of Laser Phonomicrosurgery using a Medical Robotic System

Abstract—The medical robotic system described herein has previously demonstrated promise that as a clinical tool it has the potential to improve operative outcomes by enabling increased surgical accuracy. A second area of focus is efficiency. Efficiency improvements may be realized by automating certain parts of clinical procedures. This paper explores how a laser phonomicrosurgery procedure can be automated once an excision path has been derived. The paper goes on to evaluate the potential benefits of automating such a procedure. Gradient Vector Flow [6] (GVF) based active contours were found to be an efficient means of identifying an excision perimeter. Optimal GVF parameters were determined as was a manual adjustment mechanism to address target scaleability issues. Initial accuracy and efficiency improvements indicate the approach has promise as a clinical tool.

I. INTRODUCTION

Previously the current state of the art in laser phonomicrosurgery was explored, see Giallo *et al.* [1] and an opportunity to create an improved Human Computer Interface (HCI) was identified. Subsequently, a proof of concept prototype for a medical robotic system was designed and fabricated [2]. Initial usability and accuracy experiments of the appliance were conducted and the results of these experiments were found to be promising [3]. These results led the authors to conclude that further automation of the HCI will enhance the clinical utility of this system. This paper explores the design and implementation of an automation enhancement for this medical robotic system.

In considering the concept of automation it is useful to consider first how such automation will mesh with current clinical practice in the art. In this context any automation enhancement must improve efficiency, accuracy, or patient safety in order to be of value. Toward this end the authors chose the task of tumor perimeter definition for automation. This task requires the clinician to subjectively make a determination regarding the boundaries of the tissue to be excised based upon a visual examination of the operative field. In the case of excising a malignancy this process is augmented by real time pathology to ensure adequate surgical margins are achieved. For the proposed automation enhancement to the medical robotic system the methodology employed was analysis of sample tumor images using 2D image processing to identify the perimeter of

the tissue to be excised. Essentially this is an edge detection problem and was approached as such.

MATLAB™[10] version 7.0 has been chosen as the image processing application of choice, because of: (1) the availability of the image processing software toolbox, and (2) the generic nature of the software allows custom filters to be written as required. The images of vocal fold tumors were obtained from the University of North Carolina at Chapel Hill medical school[4]. A range of images was analyzed with the objective of having increasingly complex identification issues on a per image basis. The analysis began with the simpler problems (e.g. Fig. 1) and progressed to the more complex (e.g. Fig. 11) as understanding was gained.

Fig. 1. Discrete vocal fold tumor[4]

[10] MathWorks, Natick, MA

A number of edge detection algorithms were investigated as part of this effort. The initial investigation was performed using kernel-based techniques including Canny, Sobel, Prewitt, Roberts, LaPlacian of Gaussian, and zero crossing. Comparisons of these classic edge detection algorithms were then made to determine their relative effectiveness in terms of tumor boundary detection and efficiency versus one another. The kernel-based techniques are basically either gradient-based or LaPlacian-based. In this case the Canny, Sobel, Prewitt, and Roberts methods are all gradient-based. Each of them computes a first derivative, or gradient, of the intensity of the image and then seeks to find maxima and minima within that result. The LaPlacian techniques, by contrast, seek to find maxima and minima within the second derivative of the intensity function of an image.

II. KERNEL-BASED IMAGE ANALYSIS TECHNIQUES

The Canny edge detection technique [5] was employed on the simple tumor case shown in Fig. 1 above. The RGB image of Fig. 1 was imported into MATLAB™, converted to grayscale, and then analyzed using the Canny edge detection technique. The result of the analysis is shown in Fig. 2.

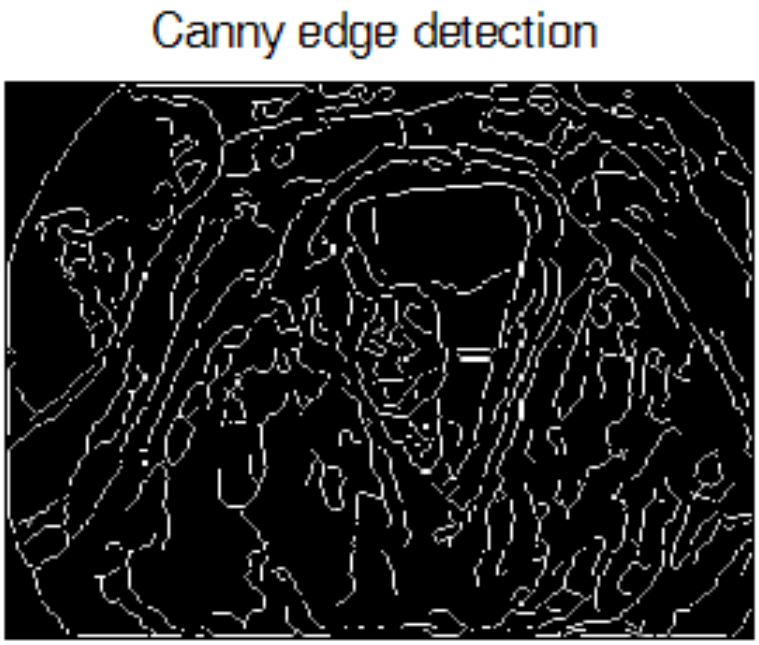


Fig. 2. Canny Edge Detection results - simple case

We observe the non-specific nature of the results and the large number of unwanted artifacts in the output image. This technique does a reasonable job of defining the tumor edges; however, the tumor characteristics that enable this definition are unfortunately not unique within the image. As a result the technique burdens the user with the need to remove the extraneous artifacts in order to make the output useful in this application.

The next approach to be evaluated was the Sobel edge detection technique. As in the preceding analysis, the simple tumor case of Fig. 1 was imported into MATLAB™

and the RGB image converted into grayscale for processing. The result of that analysis is shown in Fig. 3.

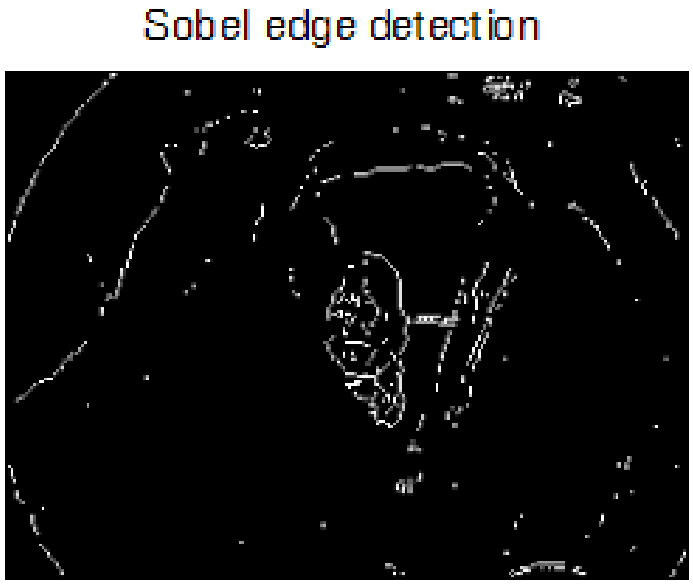


Fig. 3. Sobel Edge Detection results - simple case

We observe that the Sobel technique is a significant improvement over the Canny technique. Not only does it perform a reasonable job of outlining the tumor mass itself but it also greatly reduces the additional unwanted artifacts that are detected with the Canny algorithm. There would still be considerable post processing effort involved in translating this result into data suitable for use in guiding a surgical laser, however, it is clearly a smaller effort than that which would be associated with the Canny algorithm output.

The next edge detection technique to be evaluated was the Prewitt algorithm. This algorithm enables the user to manually set a detection threshold prior to running an analysis. This allows one to determine an acceptable compromise between the quality of discerning the tumor outline versus generating unwanted additional artifacts. As before a similar process was followed relative to the importation of a simple baseline image and converting said image into a grayscale representation.

The output generated by a MATLAB™ analysis using four different detection thresholds is shown in Fig. 4.

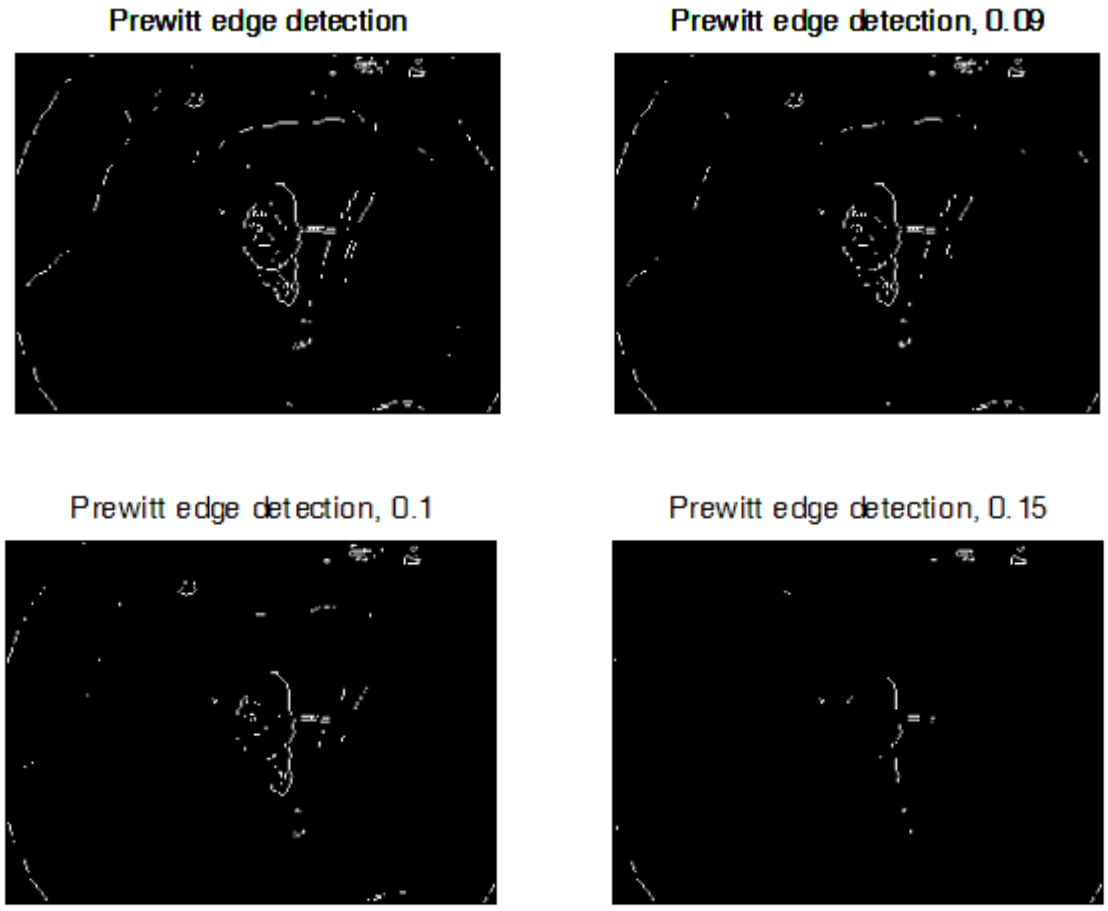


Fig. 4. Prewitt Edge Detection results - simple case

We observe that the Prewitt algorithm generates fewer unwanted artifacts than either the Canny or Sobel techniques; however, undesirable artifacts are not completely eliminated. Moreover, an increasing amount of the tumor edge detail is lost as the

detection threshold is increased. At the detection threshold levels low enough to capture adequate tumor edge detail there are still significant unwanted artifacts. It appears that there is at best a slight advantage to the Prewitt technique versus the other approaches previously analyzed.

The next edge detection technique to be evaluated was the Roberts algorithm. As with the Prewitt algorithm this technique also enables the user to manually set a detection threshold prior to running an analysis. As before a similar process was followed relative to the importation of a simple baseline image and converting said image into a grayscale representation. The output generated by a MATLAB™ analysis using four different detection thresholds is shown in Fig. 5.

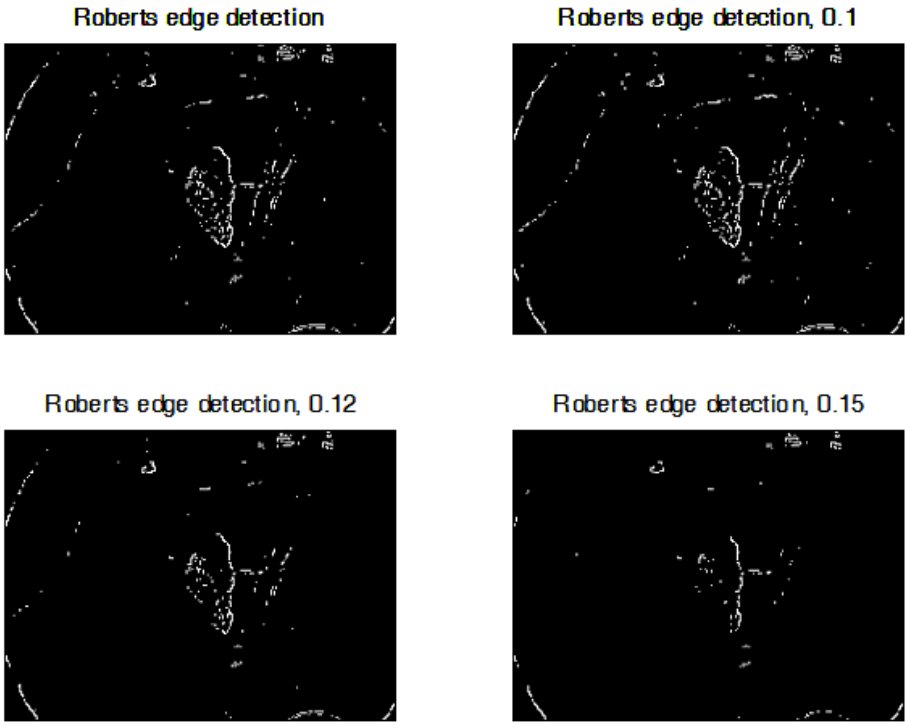


Fig. 5. Roberts Edge Detection results - simple case

Again we observe that the algorithm does a reasonable job of determining the tumor edge boundaries providing that the detection threshold is set appropriately low. As in the Prewitt algorithm we observe that as the detection threshold increases the crucial data delineating the tumor boundary is lost. Given the nature of the desired application we are again forced to conclude that this edge detection algorithm is not an appropriate choice.

LaPlacian techniques determine the edge points of an image by computing the second derivative of the image intensity. Since this calculation is noise sensitive the algorithm first applies Gaussian filtering to the image prior to making the calculation. The filtering decreases the likelihood of noise impairment by smoothing the image. The detection criterion is the presence of a zero crossing in the second derivative calculation providing that a corresponding significant peak occurred in the first derivative calculation. In Fig. 6 we observe the results of applying this technique to the simple tumor case previously described:

Fig. 6. LaPlacian of Gaussian Edge Detection results - simple case

Once again we observe that the algorithm does a good job of identifying the tumor perimeter. Unfortunately it also identifies a number of unwanted artifacts in the process and, absent specific knowledge about the tumor location in the image; it is not possible to separate the tumor information from these artifacts.

The last of the kernel-based edge detection techniques to be evaluated was the zero crossing detection algorithm. This algorithm examines the LaPlacian operator of the preceding technique to determine where zero crossings occur. Again the methodology employed was to import the simple tumor case of Fig. 1 and convert the color RGB representation into a grey scale image. The MATLAB™ results of this approach are shown in Fig. 7.

Fig. 7. Zero Crossing Edge detection results - simple case

All of these two dimensional edge detection techniques were found lacking in this particular application due to the non-unique nature of the tumor boundaries. The detection algorithms were able to differentiate the area of interest from the surrounding tissue with varying degrees of success. Unfortunately, even in the best differentiated cases the algorithms produced other artifacts in the processed image that are difficult to eliminate. One solution to this problem will be to specify an area of interest within an image prior to performing any analysis. This approach will effectively eliminate a large portion of the target image from analysis and thereby avoid production of the undesirable artifacts. Another issue prevalent in all of the foregoing techniques was the detection of

edges, or artifacts, within the confines of the tumor itself. This creates the problem of differentiating the border of the tumor from the artifacts generated within the mass itself. As a result all of these edge detection image processing methods were deemed unsuitable for guiding an autonomous excision process.

III. CONTOUR-BASED IMAGE ANALYSIS TECHNIQUES

The next class of techniques to be investigated was that of contours. Contours are regions of constant intensity within an image. This was felt to be a more promising approach due to the increased ability of these techniques to identify a non-uniform perimeter. Two types of contour techniques were investigated: constant intensity contours and active contours, also known as snakes.

1) CONSTANT INTENSITY CONTOURS

The methodology followed for this analysis was similar to that employed for the kernel-based edge detection methods. The simple tumor case of Fig. 1 was imported into MATLAB™ as a *.jpeg image file. Software code was then written to create families of contours within the sample image. Contour plots were created within ranges using a fixed increment. For example, the following contours have the parameter of 100:10:200 which yields contours with intensities between 100 and 200 incrementing by 10 intensity points for each contour. The RGB image was filtered with several techniques (listed above each plot) and then the contour plots of Fig. 8 were created.

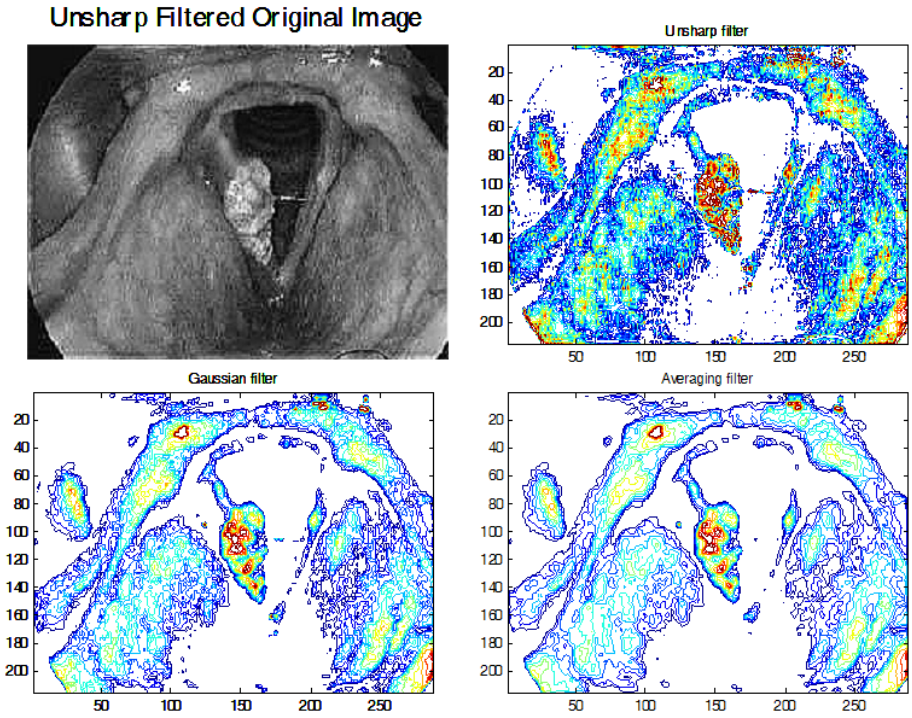


Fig. 8. Constant intensity contours example – simple tumor case

Additional contours were also created using the Sobel and LaPlacian of Gaussian filtering techniques but the results were completely unusable. From the plots above we can see that the tumor is clearly identified regardless of the applied filtering, however, the intensities that allow us to identify the tumor boundaries also appear in other areas of the image. This results in the generation of artifacts that must be eliminated in order to make use of this data in an autonomous process. With both the kernel-based edge detection methods and the constant intensity contour approach it is increasingly clear that there is a

need to specify a region of interest for the algorithm to operate upon in order to reduce the incidence of undesirable artifacts.

2) ACTIVE CONTOURS

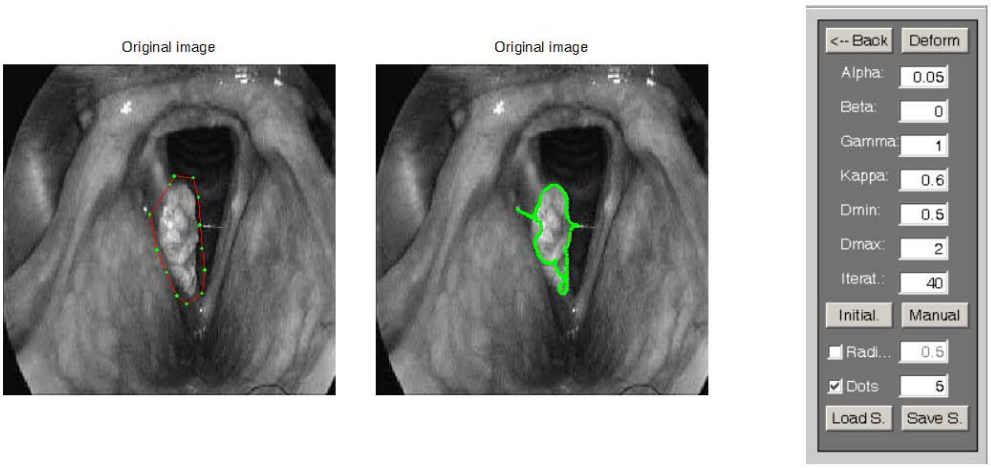
Active contours are computer-generated curves that move in order to determine the perimeter of an object located within an image. Active contours are so named because the algorithm determines the edges of an object within an image as a function of both internal and external forcing functions. This property differentiates the technique from passive computational approaches such as the previously discussed kernel methods. Active contours are also known as snakes. This is due to the convergence algorithm employed in the technique that makes the movement of the contour appear "snakelike" as it reaches its endpoint. Active contours have been used in many applications for the purpose of determining object boundaries. Historically there have been two major shortcomings associated with this technique. First, the earlier generation implementations of this algorithm were very sensitive to the starting position or "region of interest". This "region of interest" is an initial perimeter surrounding the object whose exact boundaries we would like the active contour to determine. With the older implementations of the algorithm the final convergence position of the contour was highly dependent upon the definition of the region of interest. If the region was too large, often the contour would not converge to the object boundaries. The next significant

drawback to the earlier active contour implementations was their inability to conform to indentations, or concavities, present within the perimeter of the object. As a result the converged contour did not faithfully describe the true object outline.

The work of researchers Xu and Prince[6] has attempted to mitigate these deficiencies. Their research has focused on parametric active contours and more particularly on the external forcing functions of these contours. They developed a new external forcing function, called Gradient Vector Flow (GVF) that claims to remedy the two classic deficiencies of active contours. By minimizing an energy function associated with the GVF field definition, Xu and Prince realize improved performance in the areas of initialization sensitivity as well as conformance to object boundary concavities.

In this application we wish to precisely define the perimeter of an area of interest in order to surgically excise it. A number of experiments were conducted using MATLAB™ software for GVF active contours developed by researchers at Johns Hopkins University[11]. Various tumor images were analyzed in order to refine the GVF function parameters. It is expected that proper choices for the function parameters will significantly affect the accuracy of the final result. The initial results of the analysis effort for the simple tumor case are shown Fig. 9.

[11] http://iacl.ece.jhu.edu/projects/gvf/



• Low rigidity
• Moderate external force weight

Fig. 9. GVF snake analysis - simple tumor case

In Fig. 9 we observe the original and now familiar grey scale representation of the simple tumor of Fig. 1 with a "region of interest" defined on the image. This region of interest corresponds to the initialization position selected by the user for the active contour. The green outline around the tumor in the right side picture shows the snake in its final convergence position. The GVF parameters for this experiment are summarized on the right[12]. Not unexpectedly we observe that the convergence properties of the contour are highly influenced by the parameter selection choices. Many additional

[12] Alpha is the tension of the snake, Beta is the rigidity of the snake, Gamma is the step size in one iteration, Kappa is the external force weight, Dmin is the minimum resolution of the snake and Dmax is the maximum resolution of the snake.

experiments were performed to refine and optimize these parameter choices. The end result of this refinement effort is summarized in Fig. 10.

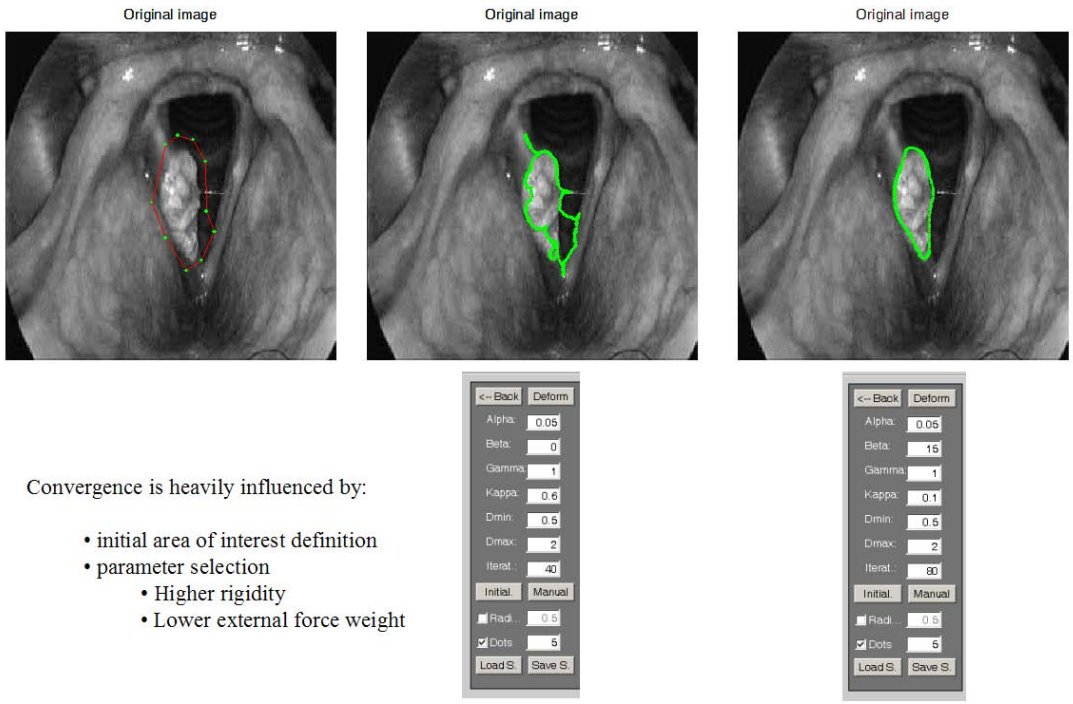


Fig. 10. Refined GVF snake analysis - simple tumor case

In Fig. 10 we observe several characteristics of interest. The frame on the left shows the grey scale image of the filtered image with a fairly poor (i.e. excessively large and loosely defined) initialization perimeter. The center frame shows the poor contour convergence that resulted from employing the default software parameters. The frame on the right shows an excellent convergence result that is directly due to the refined

parameter choices for the GVF function. It is clear that despite the relatively poor initialization perimeter that was chosen the contour is infinitely more sensitive to parameter selection values than it is to the initialization perimeter definition.

Based upon the promising results achieved above with a simple case there was sufficient confidence in the active contour approach to evaluate the performance of this technique with more complex cases. The methodology employed for the complex case was identical to that of the simple case. An image of interest was imported into MATLAB™ and converted into a grey scale representation. MATLAB™ software was again used to evaluate the effects of (1) GVF parameter choices and, (2) the definition of the initialization perimeter on the ultimate convergence of the active contour solution.

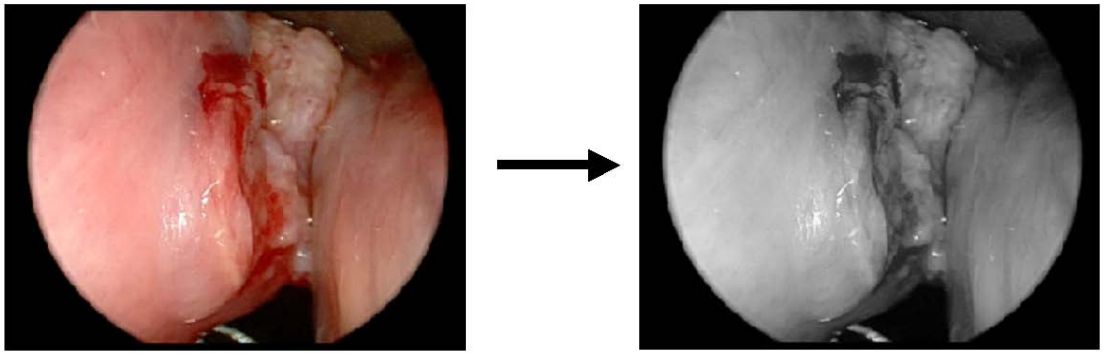


Fig. 11. Image preparation - complex tumor case[4]

We observe the significantly more difficult characteristics of this case, notably the indefinite perimeter definition that would challenge a skilled clinician to accurately determine the appropriate excision boundaries. Next, the GVF software was employed to determine the proposed perimeter.

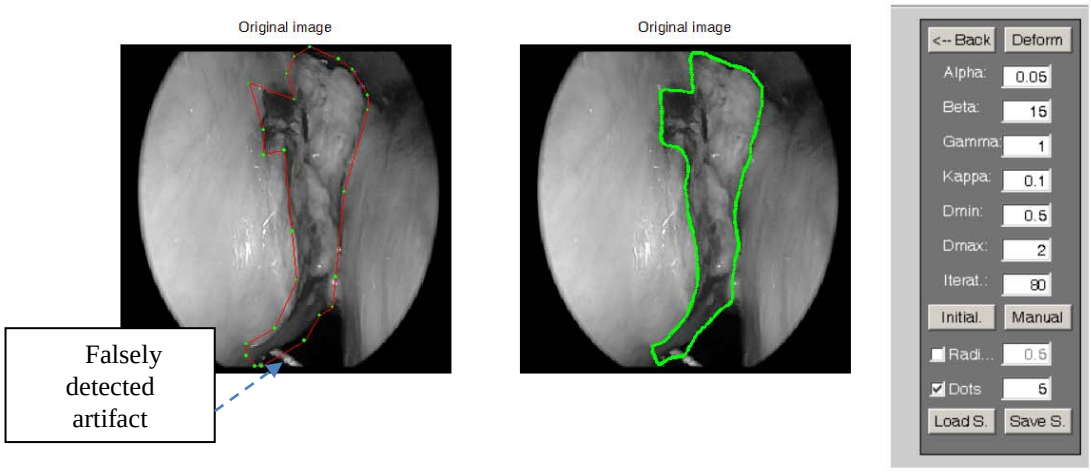


Fig. 12. Refined GVF snake analysis - complex tumor case

The left frame of Fig. 12 depicts the grey scale representation of the tumor with the initialization perimeter defined. The chosen perimeter is fairly close to the perceived tumor boundary and is a reasonable definition given the visual cues present in the image. In the right frame we observe the end result after the contour has converged to the calculated tumor boundary. Using the same refined parameters as were employed in the simple tumor case we observe similar good performance. The contour faithfully traces

the actual tumor boundary with very few deviations. The main issue appears to be where the dark area of the open airway has an artifact falsely detected by the algorithm as an edge. In practice this would likely not pose a safety issue as the airway adjacent to the tumor is blocked with saline soaked cotton pledgets prior to the use of the surgical laser. The pledgets will absorb the energy of any errant laser shots to preclude the possibility of an airway fire.

To begin to address the challenges outlined in the research proposal several initial experiments have been conducted. Building upon the image processing analysis that was previously performed several new metrics have been introduced. As described in the section detailing the challenges of registration, images have been analyzed in 2D to compute a bounding box, centroid, and tumor area. At this time the computations pertain to a user defined region of interest. They do not yet refer to the precise outline and area of the tumor but that is the logical next step of this research. For these experiments the now familiar methodology of utilizing MATLAB™ as the image processing tool was chosen. The simple case of Fig. 1 was first examined. The image was imported into MATLAB™ and then analyzed using software specifically developed for this purpose. Fig. 13 captures the initial image, intermediate stage analysis, and final result.

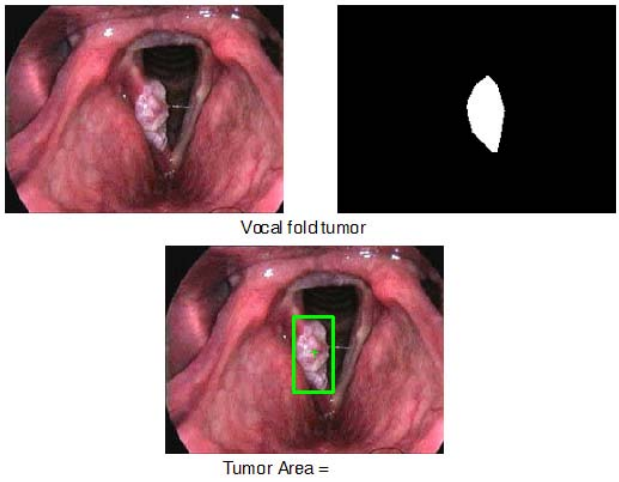


Fig. 13. Tumor Bounding Box, Centroid, and Area

The panel on the left shows the original image after importation into MATLAB™. The right side panel shows a binary representation of the user defined area of interest. The bottom panel shows the final result with a bounding box drawn that completely encompasses the tumor as well as a computed centroid (indicated by the "+") for the region of interest. Providing that the operator is careful in his/her definition of the region of interest, then this result will closely approximate the centroid of the actual tumor as well.

IV. IMPLEMENTATION OF AUTOMATION

It is clear from the foregoing analysis that the use of active contours to define tumor boundaries for phonomicrosurgery is very promising. The GVF enhanced active contours with appropriate forcing function parameters provide superior performance to

both the kernel-based image processing techniques that were evaluated as well as several types of contour analysis. The objective of the investigation and the conducted experiments was to determine if an automated tumor perimeter definition system could be developed that would approach the quality level that would be delivered by an experienced clinician. It appears that in the case of discrete, well defined tumors this technique may produce results comparable to a clinician defined excision perimeter. However, as the complexity of the tumor increases and its boundaries are less clear, further investigation of contour image processing techniques is needed to determine if an automated system could consistently deliver results on par or better than those achieved using classic visual identification of tumor boundaries.

The next step in this research was to employ the GVF techniques previously described to convert the region of interest perimeter into the exact tumor perimeter. MATLAB™ software code was developed for this purpose. An initial outline around the area to be excised is created per Fig. 14.

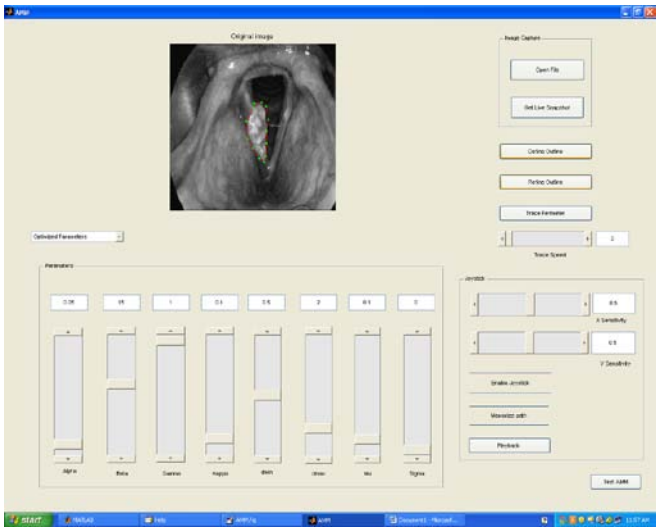


Fig. 14. Sample excision outline

The GVF algorithm is then invoked to refine the initial outline into a final excision vector. The result is shown in Fig. 15.

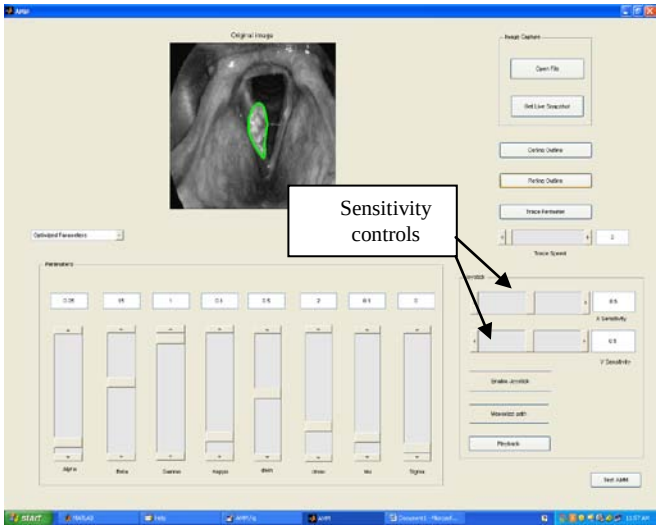


Fig. 15. Final excision outline

The first issue to be addressed is that of scaleability. The human adult vocal fold typically ranges in size from 17-21mm in males and 11-15mm in females [7]. The challenge is to create an excision vector that is appropriately sized based upon the final active contour vector. This is complicated by the fact that phonomicrosurgery is typically done under high magnification to facilitate visualization. The current GUI software is designed to accept either captured image files such as are seen in this paper or to capture a live snapshot of the operative field. In either case the captured image is significantly larger than the actual operative field. As a result, the active contour analysis will create the appropriate shape for the excision vector but it will not be the correct size.

To address this challenge it is necessary to match the contour vector to the operative field. The software does this by first scaling the excision vector to the workspace of the medical robotic system designed and developed for this task. The sensitivity controls (see Fig. 15) are then used to refine the excision vector to the appropriate scale for the actual tumor size. The effect of these controls on the excision vector is shown in Fig. 16. The leftmost figure shows the initial excision vector before optimization. The rightmost figure shows the final result after using the sensitivity controls to optimize the vector. The center figure shows the results of repeated laser runs over the initial non-optimized vector. In a clinical setting the protocol would be to adjust the sensitivity controls to optimize the excision vector while observing the effect on the

HeNe aiming beam. Once the clinician is satisfied with the optimized aiming beam trace the laser can then be employed to perform the actual excision in an automated or semi-automated procedure.

Fig. 16. Sample excision vector outlines

Additional software development has also been written to automate ablation of an entire excision area. In this case the optimized excision vector is obtained as described previously. The master control program then uses this information to create a scanning pattern of the entire excision area. The appliance then, on command, will ablate the area in a "line by line" fashion. Photographs of the final result were not particularly helpful as the repeated laser firing, even at low power settings, tended to distort the Zap-It™ paper (see center frame of Fig. 16). Observations of the ablation process confirmed that the system appears to faithfully track the desired region. Further experiments will be needed to completely validate this preliminary conclusion.

V. CONCLUSIONS AND FUTURE WORK

The use of GVF-based active contours was found to be an effective mechanism for identifying the perimeter of an excision boundary. GVF parameter selection choices were proven to have a critical effect on algorithm performance and were optimized to the application at hand. Automation of the determination of an excision perimeter was accomplished and mechanisms to manually adjust target scaleability were implemented. The medical robotic system, a clinical tool, clearly demonstrates that the potential exists for improvements in clinical accuracy and efficiency.

Future work can be compartmentalized into several categories. At a strategic level the largest opportunity is to advance the clinical state of the art with regard to accuracy and efficiency. Integrating an advanced imaging modality such as optical coherence tomography (OCT) into the operative setup has tremendous potential benefits. Employing OCT to control laser aiming would provide the clinician with the ability to ascertain in-vivo the best path for excision. OCT offers the promise of precise determination of the boundaries of malignant tissue and thus the ability to determine exact surgical margins. This will minimize vocal deficits resulting from surgery. Operative times would likewise be reduced since classic pathology and its inherent delays would be eliminated. This could potentially lead to directly reduced costs and superior clinical outcomes as time under general anesthesia is decreased.

At a system level, several improvements are envisioned. First, the open loop nature of the control system could be addressed. Further work in image processing could deliver the capability to automatically effect the scaling adjustments that are currently implemented with the manual sensitivity controls. In order to achieve this it is expected that experimental data would potentially reveal optimum settings. By conducting numerous experiments with captured tumor images and recording the manually derived sensitivity data for a faithful excision perimeter, it may be possible to create a rule for configuring the system. This would be dependent upon verification that the manually derived settings converge to constant values for the x-axis and y-axis parameters. Automatic registration of the operative image with the operative field could likewise be accomplished. In this situation it will be necessary to establish a world coordinate frame and synchronize the MRS with the operative field image.

At a tactical level the implementation of the appliance must undergo a number of refinements. First, an alternative packaging approach must be developed to transition the device into a clinical setting. A separate enclosure for the slave microcontroller circuit board in conjunction with a regulatory compliant power and grounding configuration require development. This development would also encompass an upgraded power supply design to improve heat dissipation during continuous operations. The joystick interface could be further improved by transitioning to an implementation using the fine motor skills of the fingers rather than the current gross level movements of the arm,

wrist, and hand. This requires a new, smaller joystick and corresponding mounting platform. This improvement would also provide the opportunity to incorporate the appliance controls into a more ergonomic arrangement. For example, the new smaller joystick could be incorporated into the arm of an appropriate chair for the operating theater.

In total there are many exciting avenues that future research and development could pursue.

REFERENCES

[1] J. F. Giallo, II; Edward Grant; Robert Buckmire, "An Evaluation of User Interfaces for Laser Phonomicrosurgery," 2008. Submitted to Transactions on Information Technology in BioMedicine. Reference number, TITB-00173-2008.

[2] J. F. Giallo, II; Edward Grant, "Designing of a Medical Robotic System for Laser Phonomicrosurgery," 2008. Submitted to Robotics and Autonomous Systems. Ms. Ref. No.: ROBOT-D-08-00208.

[3] J. F. Giallo, II ; Edward Grant; Robert Buckmire; Charles Finley, "The Testing and Evaluation of a Medical Robotic System for Laser Phonomicrosurgery," 2008.

[4] R. M. D. Image provided courtesy of Buckmire, "Associate Professor and Chief, Division of Voice and Swallowing Disorders, Department of Otolaryngology Head and Neck Surgery, University of North Carolina at Chapel Hill Medical School," 2006.

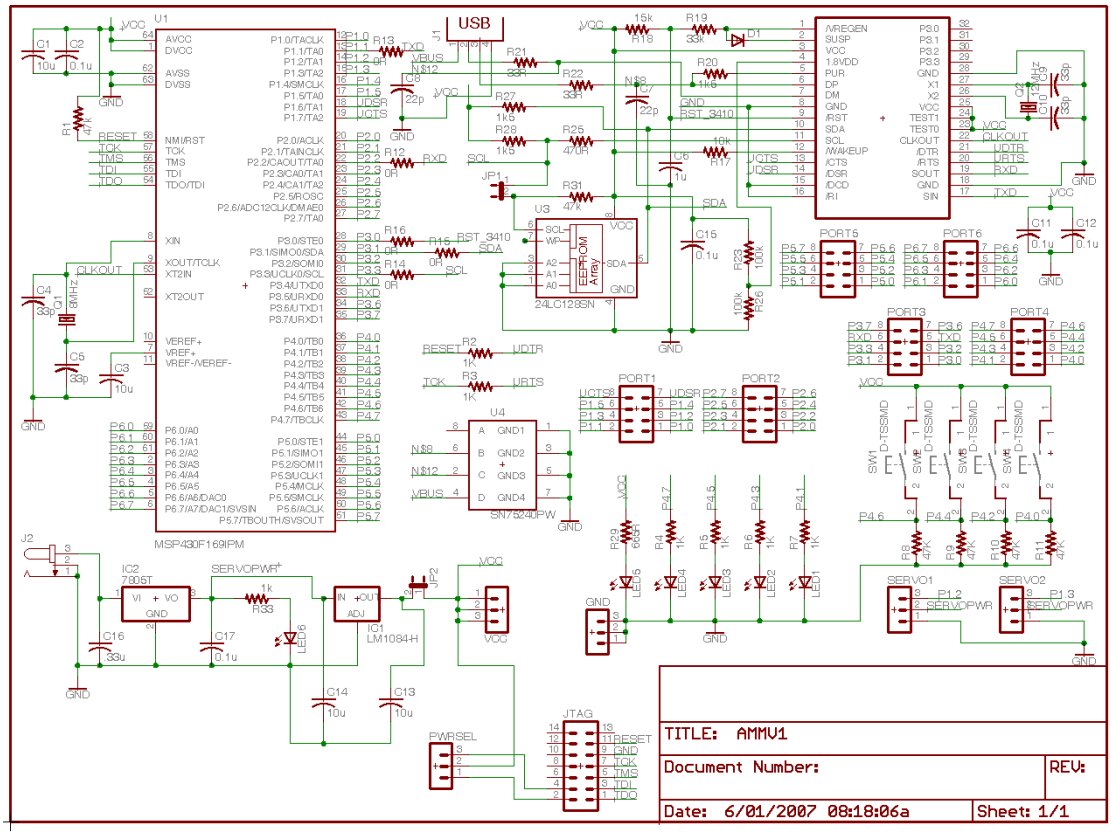
[5] J. Canny, "A Computational Approach to Edge-Detection," *Ieee Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, pp. 679-698, Nov 1986.

[6] C. Y. Xu and J. L. Prince, "Generalized gradient vector flow external forces for active contours," *Signal Processing*, vol. 71, pp. 131-139, Dec 1998.

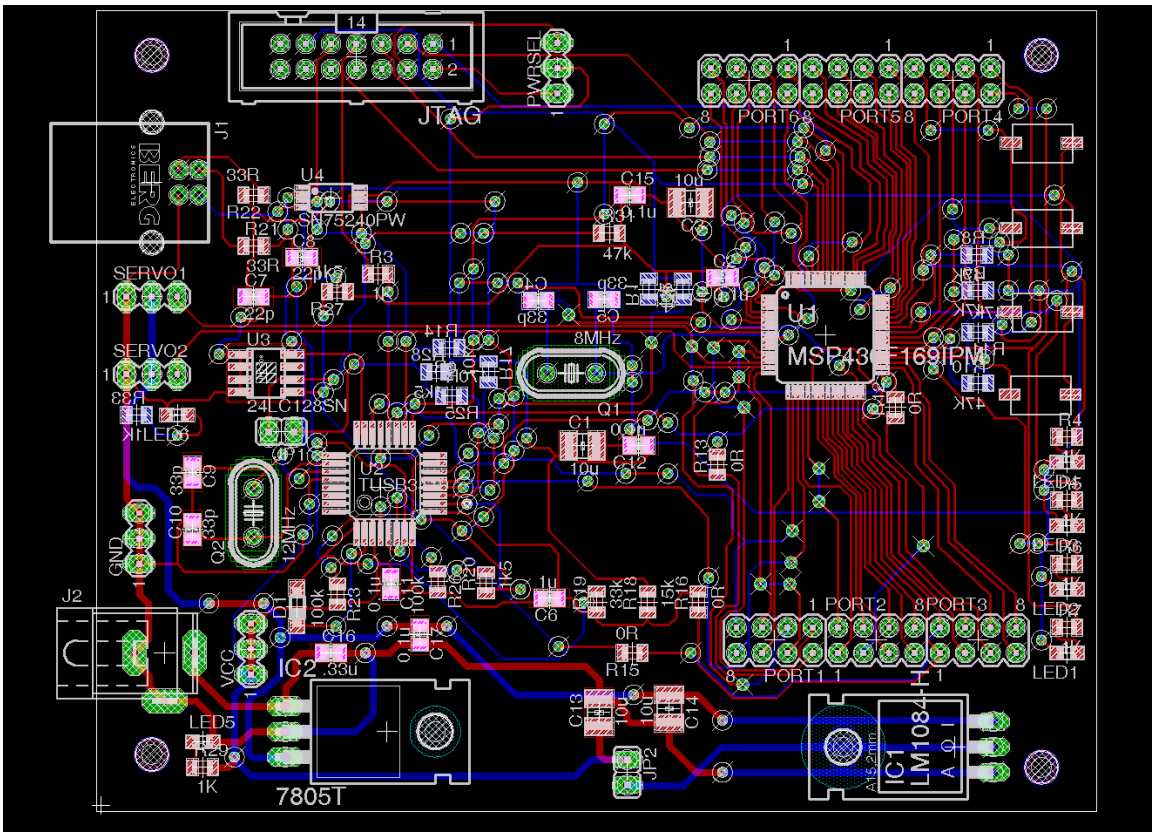
[7] D. M. Bless and J. H. Abbs, *Vocal fold physiology :contemporary research and clinical issues*. San Diego, Calif.: College-Hill Press, 1983.

Appendices

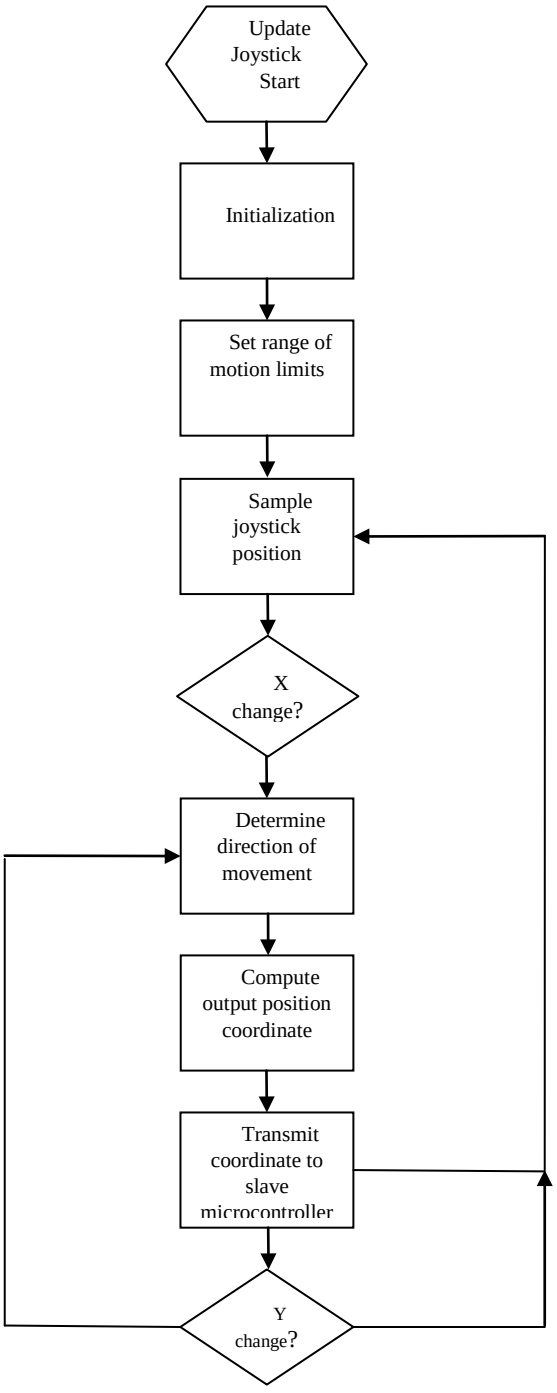
Appendix I. Hardware Circuit Schematic



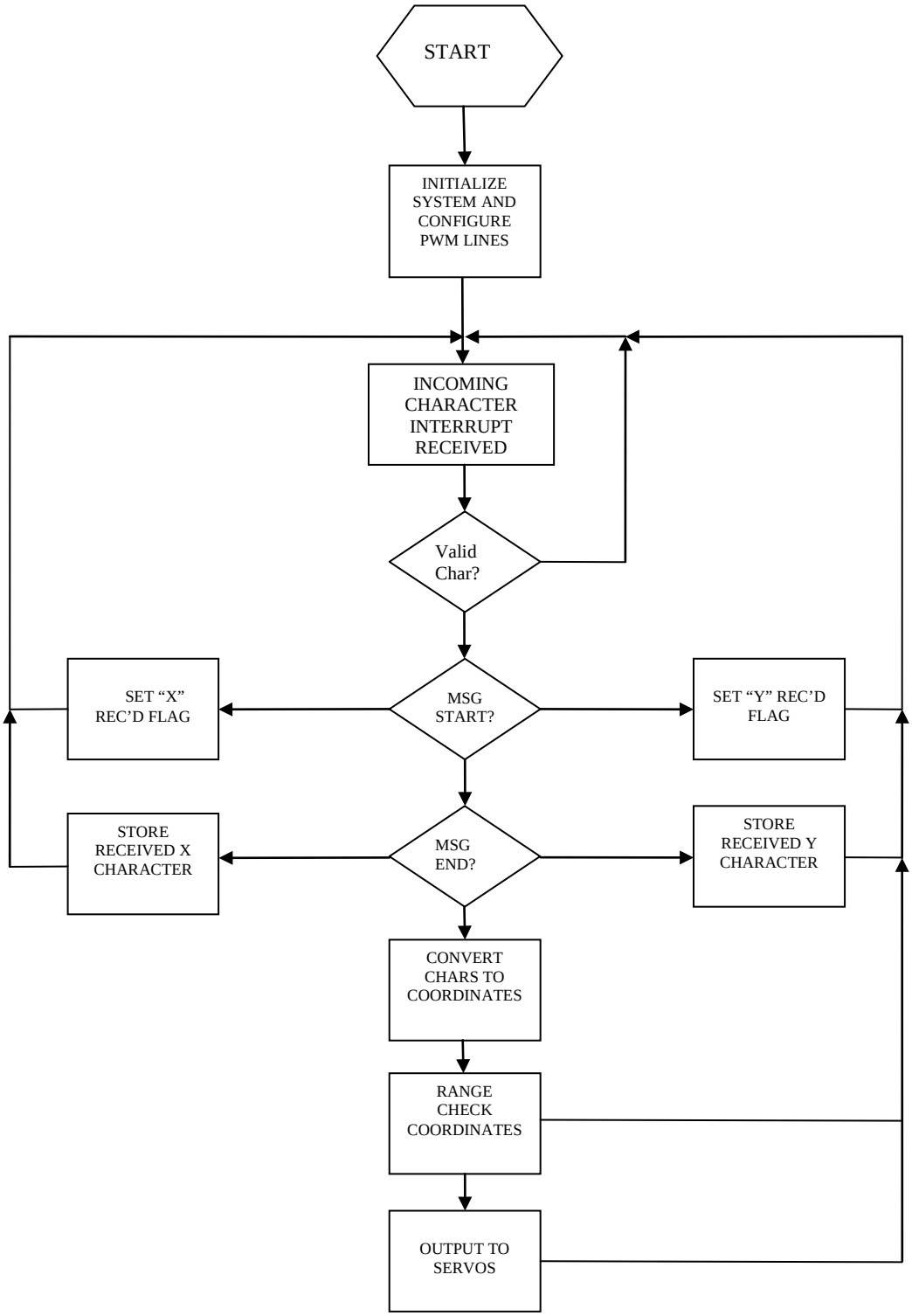
Appendix II. Circuit board layout



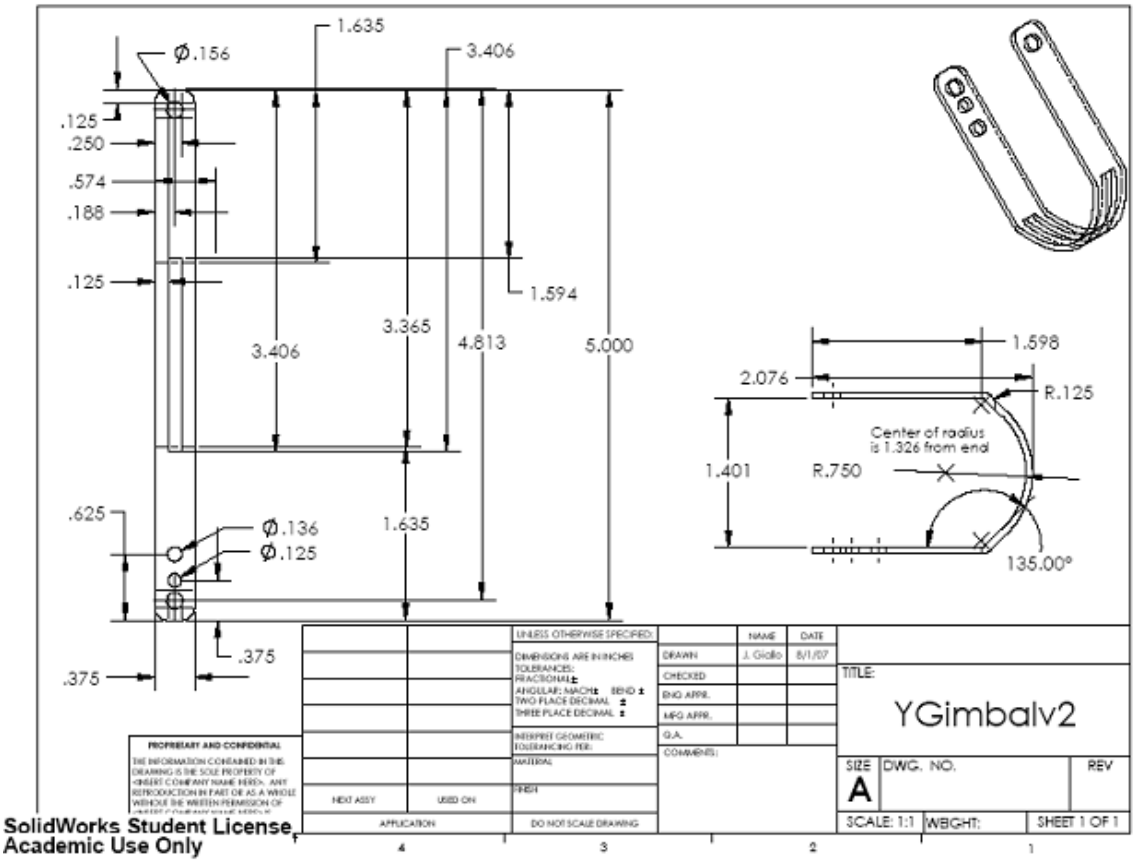
Appendix III. Master Control Program Flowchart

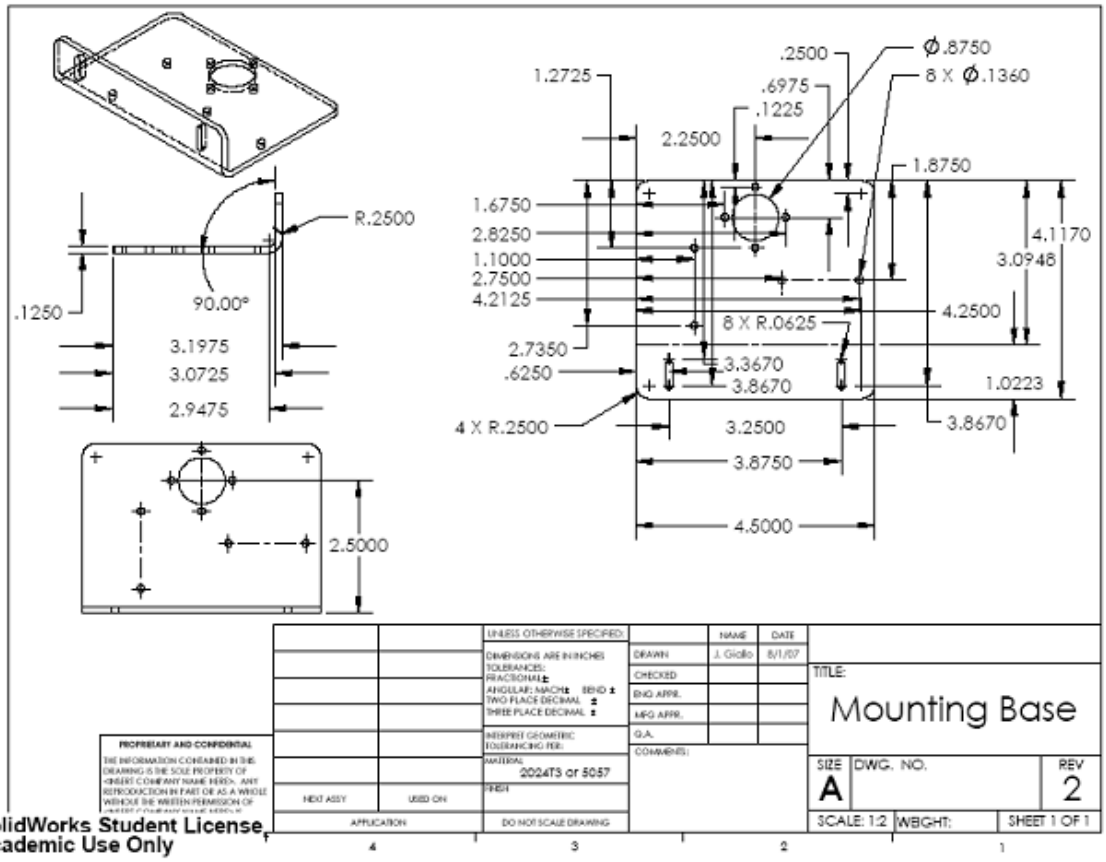


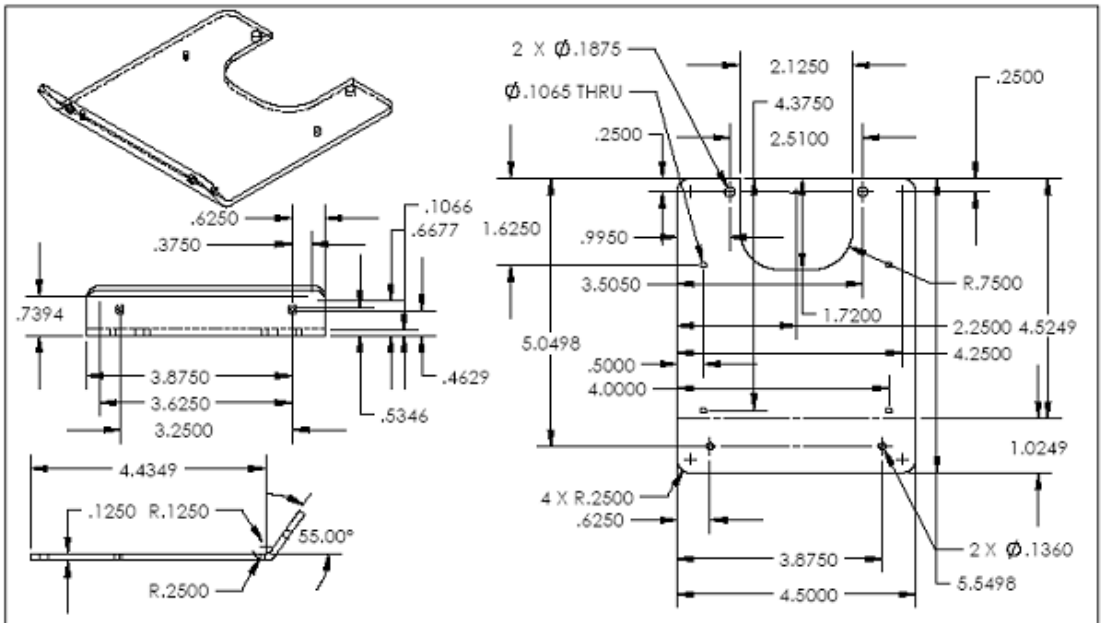
Appendix IV. Slave Microcontroller Program Flowchart



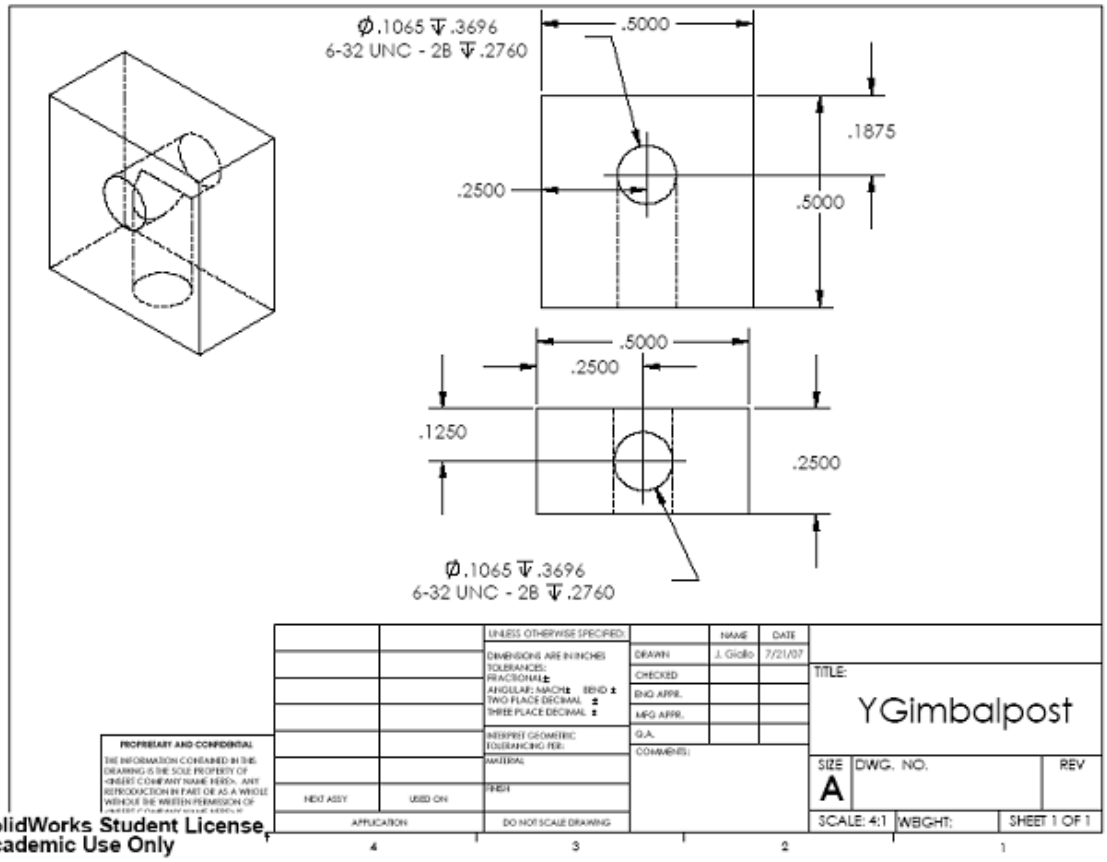
Appendix V. Mechanical drawings

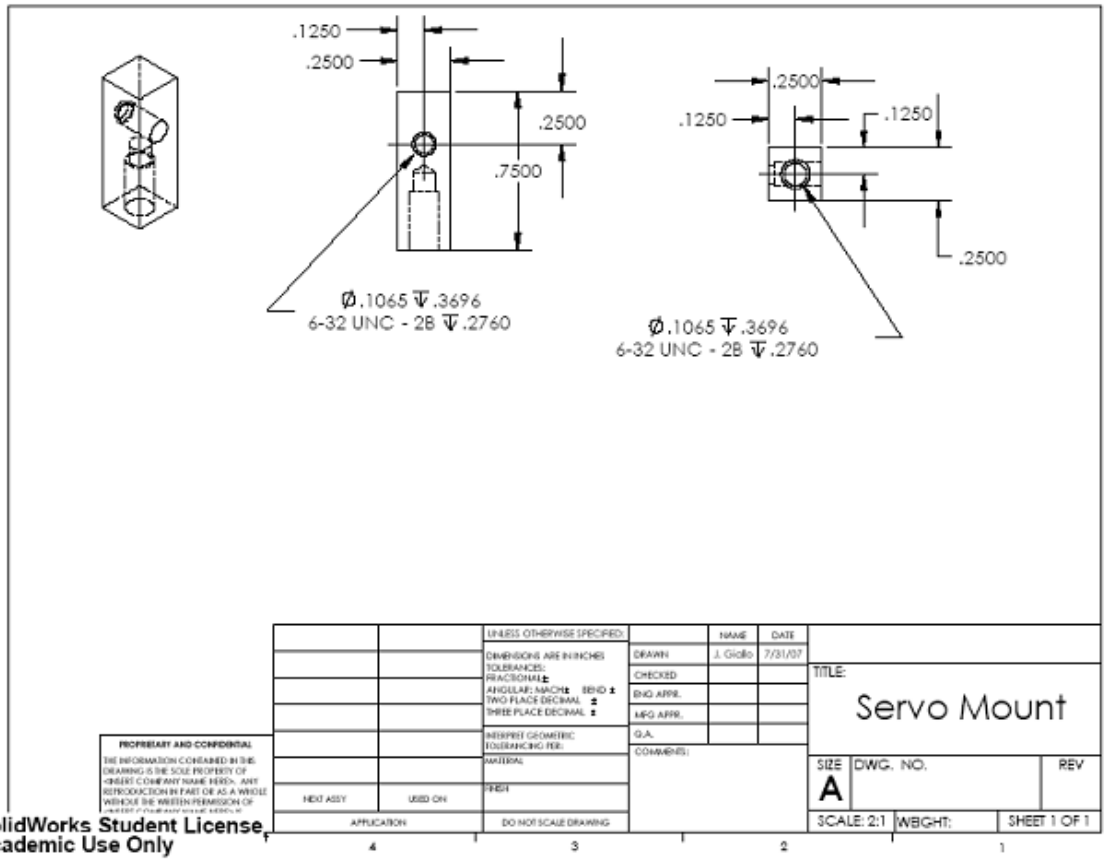


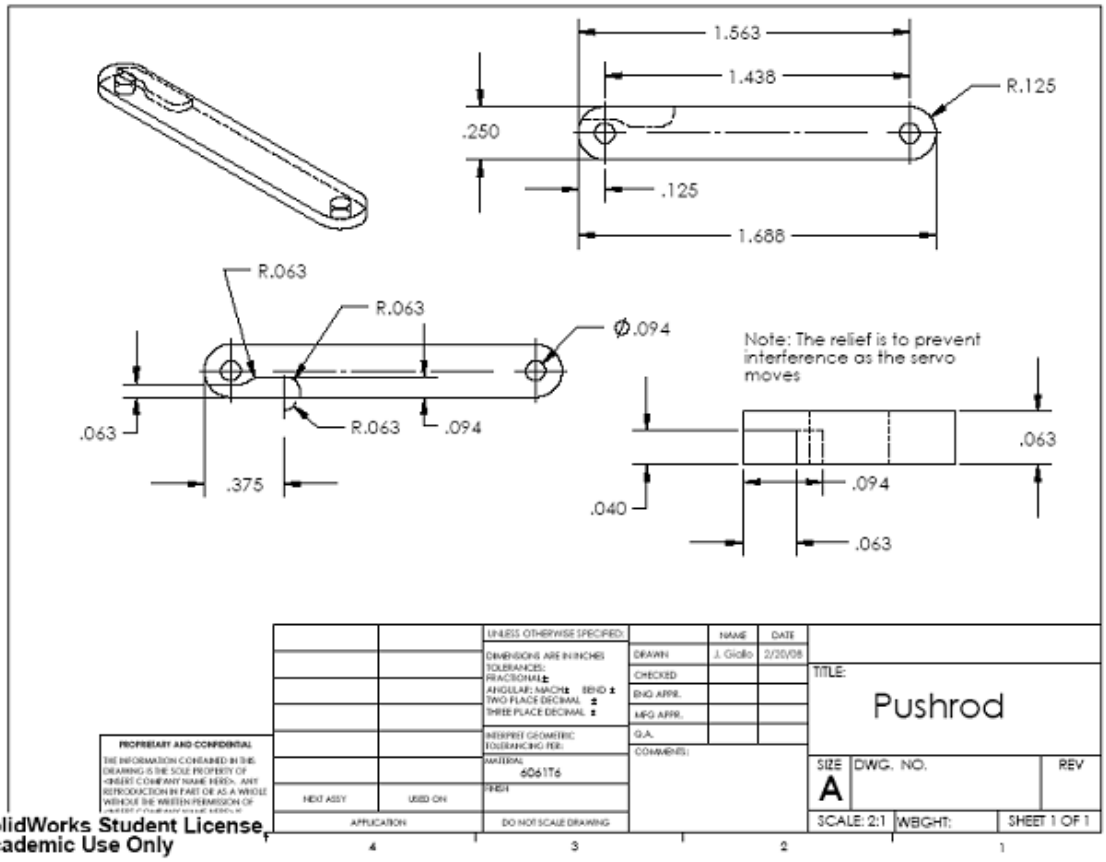

R.2500
.1250
90.00°
3.1975
3.0725
2.9475
2.5000
1.2725
.2500
Ø.8750
8 X Ø.1360
.6975
.1225
2.2500
1.8750
1.6750
2.8250
1.1000
2.7500
4.2125
4.1170
3.0948
4.2500
8 X R.0625
2.7350
.6250
3.3670
3.8670
1.0223
4 X R.2500
3.2500
3.8670
3.8750
4.5000
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
2024T3 or 5057
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
8/1/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Mounting Base
SIZE
A
DWG. NO.
REV
2
SCALE: 1:2
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License
Academic Use Only

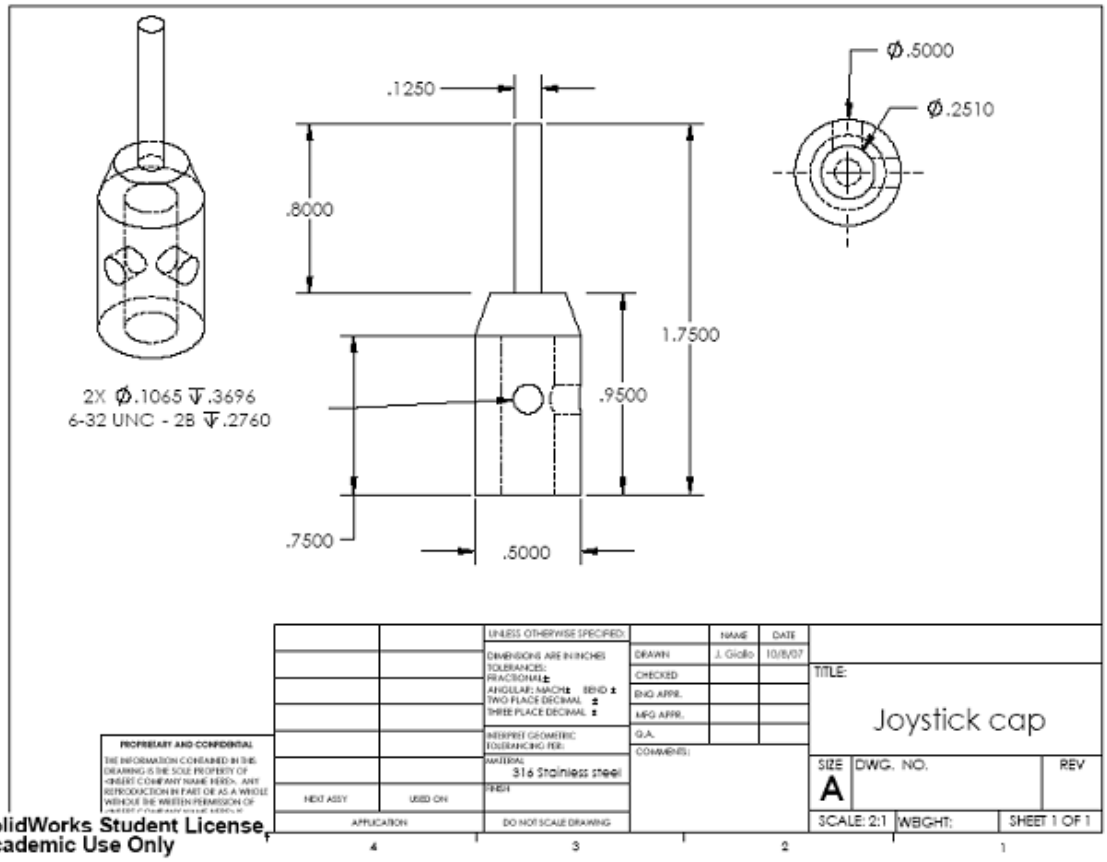


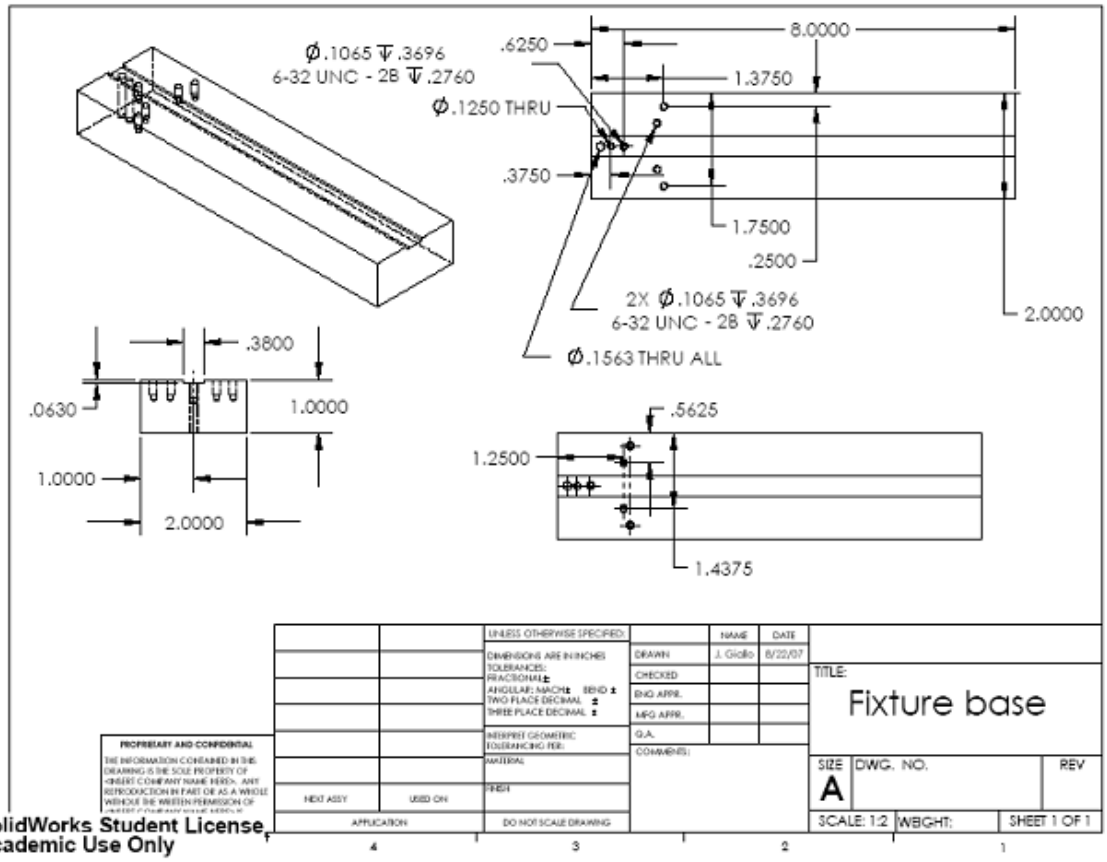
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
		DIMENSIONS ARE IN INCHES	DRAWN	J. Giallo	7/31/07	TITLE: Mounting Bracket
		TOLERANCES: FRACTIONAL ±	CHECKED			
		ANGULAR: MACH ± BEND ±	ENG APPR.			
		TWO PLACE DECIMAL ±	MFG APPR.			
		THREE PLACE DECIMAL ±	Q.A.			
		INTERPRET GEOMETRIC TOLERANCING PER:	COMMENTS:			SIZE A | DWG. NO. | REV
		MATERIAL 2024T3 or 5057				
NEXT ASSY	USED ON	FINISH				
APPLICATION		DO NOT SCALE DRAWING				SCALE: 1:2 | WEIGHT: | SHEET 1 OF 1

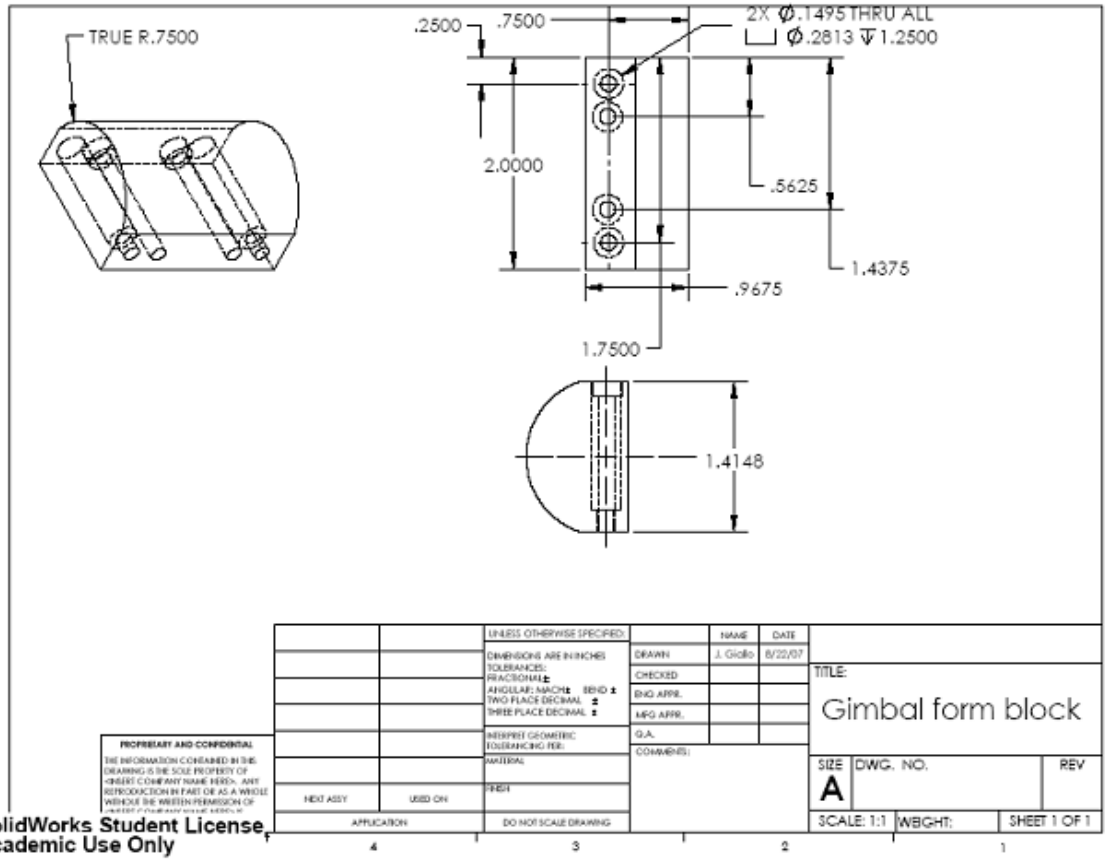

Ø.1065 ↧.3696
6-32 UNC - 2B ↧.2760
.5000
.1875
.2500
.5000
.5000
.2500
.1250
.2500
Ø.1065 ↧.3696
6-32 UNC - 2B ↧.2760
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
7/21/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
YGimbalpost
SIZE
A
DWG. NO.
REV
SCALE: 4:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License
Academic Use Only

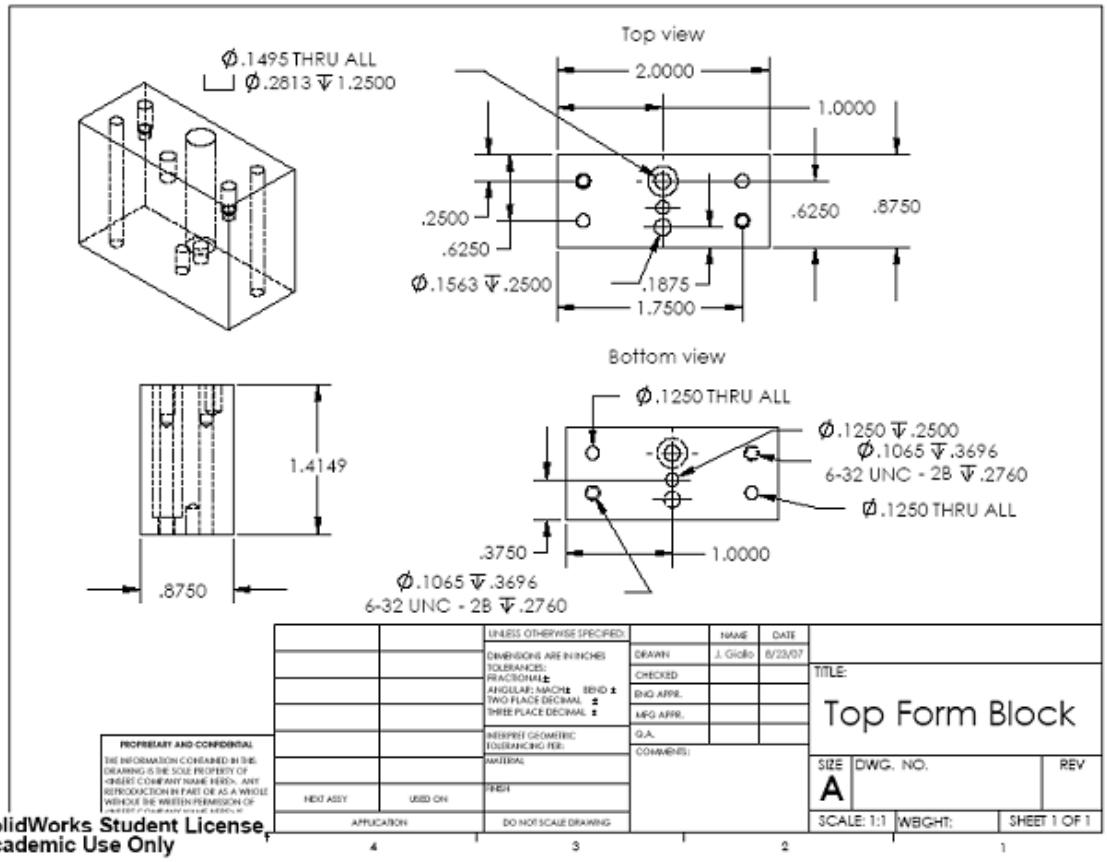

.1250
.2500
.2500
.7500
Ø.1065 ↧ .3696
6-32 UNC - 2B ↧ .2760
.2500
.1250
.1250
.2500
Ø.1065 ↧ .3696
6-32 UNC - 2B ↧ .2760
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
7/31/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Servo Mount
SIZE
A
DWG. NO.
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License
Academic Use Only

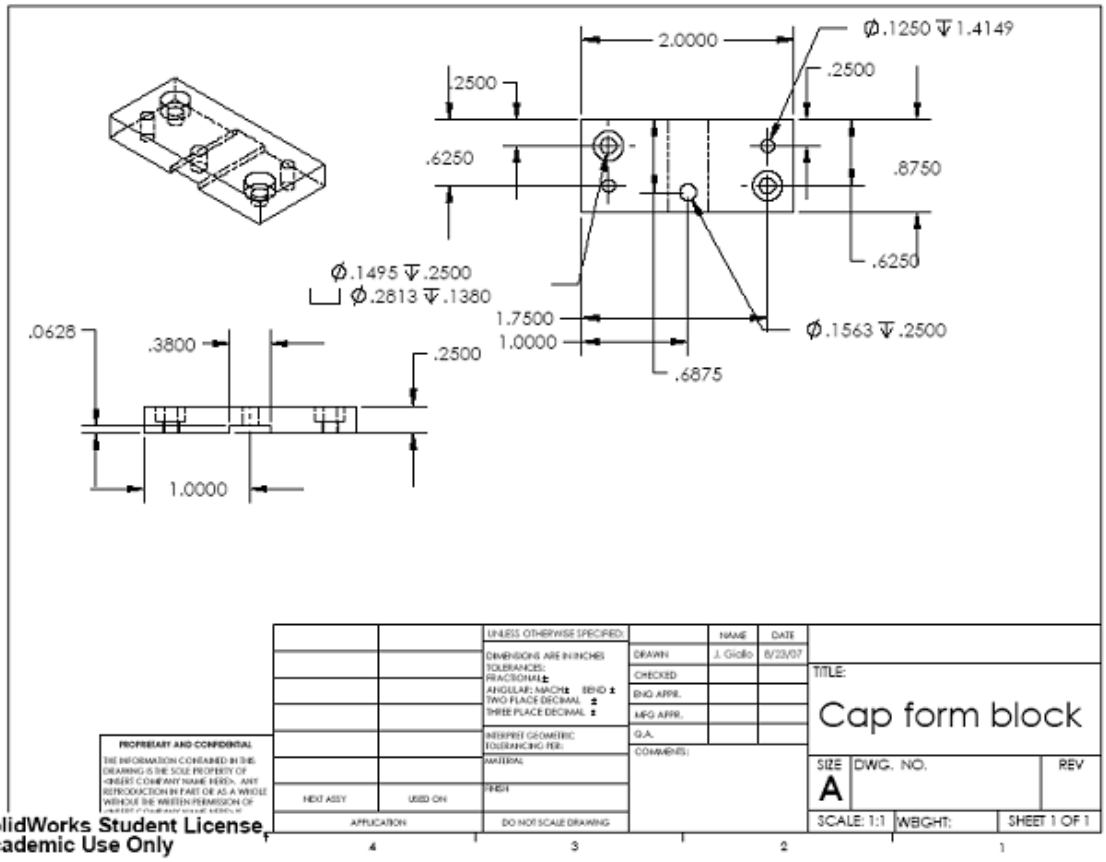


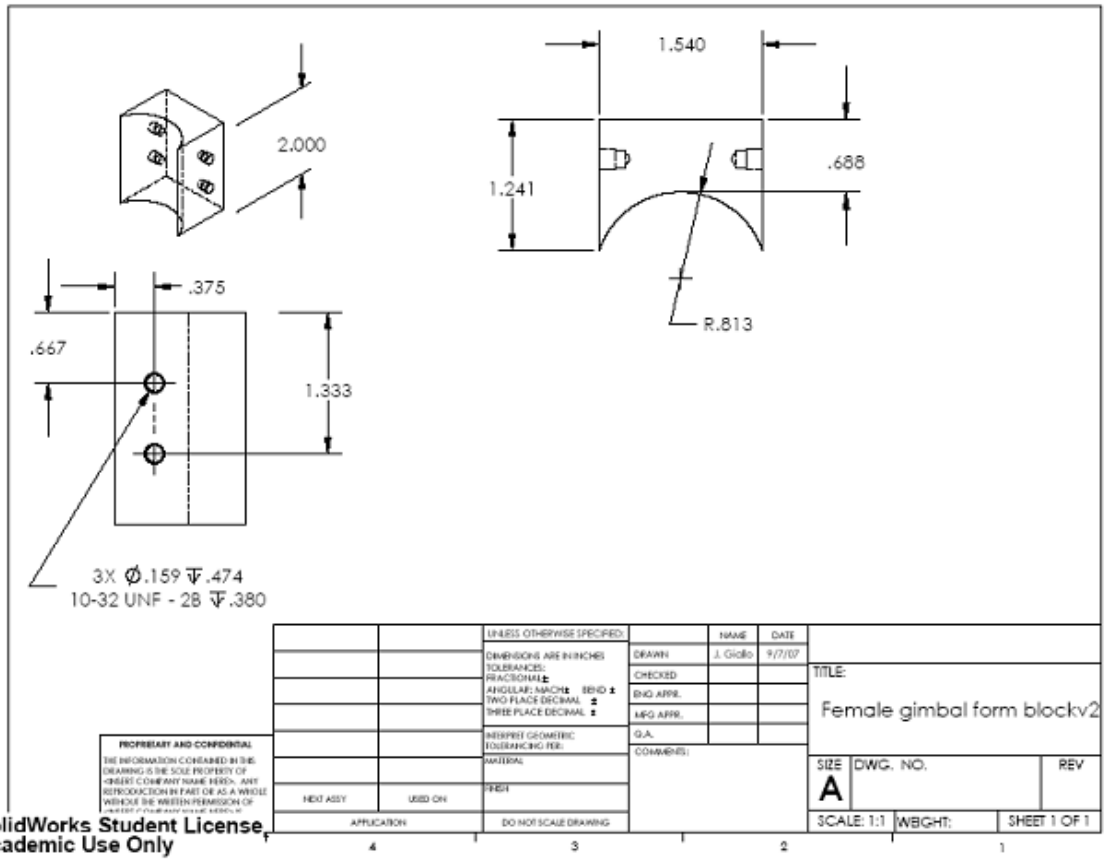

Ø.5000
Ø.2510
.1250
.8000
1.7500
.9500
.7500
.5000
2X Ø.1065 ↧.3696
6-32 UNC - 2B ↧.2760
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
316 Stainless steel
FINISH
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
DRAWN
J. Giallo
10/8/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
NAME
DATE
TITLE:
Joystick cap
SIZE
A
DWG. NO.
REV
SCALE: 2:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License,
Academic Use Only

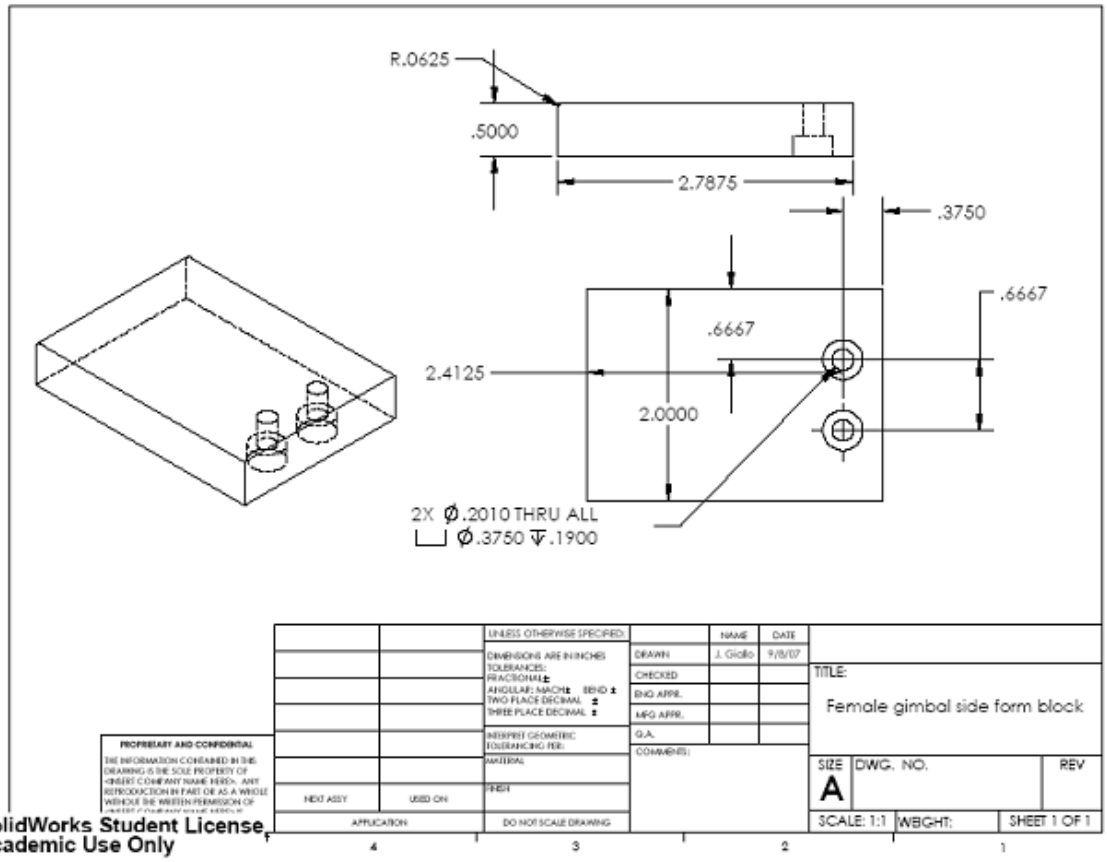

Ø.1065 ↧.3696
6-32 UNC - 28 ↧.2760
.6250
8.0000
1.3750
Ø.1250 THRU
.3750
1.7500
.2500
2.0000
2X Ø.1065 ↧.3696
6-32 UNC - 28 ↧.2760
Ø.1563 THRU ALL
.3800
.0630
1.0000
1.0000
2.0000
.5625
1.2500
1.4375
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NAME
DATE
DRAWN
J. Giallo
8/22/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Fixture base
SIZE
A
DWG. NO.
REV
SCALE: 1:2
WEIGHT:
SHEET 1 OF 1
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License,
Academic Use Only


TRUE R.7500
.2500
.7500
2X Ø.1495 THRU ALL
Ø.2813 ↧1.2500
2.0000
.5625
1.4375
.9675
1.7500
1.4148
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
8/22/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Gimbal form block
SIZE
A
DWG. NO.
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License
Academic Use Only




2.0000
Ø.1250 ↧1.4149
.2500
.2500
.6250
.8750
.6250
Ø.1495 ↧.2500
⌴ Ø.2813 ↧.1380
1.7500
1.0000
Ø.1563 ↧.2500
.6875
.0628
.3800
.2500
1.0000
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
8/23/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Cap form block
SIZE
A
DWG. NO.
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License,
Academic Use Only


1.540
2.000
1.241
.688
.375
.667
1.333
R.813
3X Ø.159 ↧.474
10-32 UNF - 28 ↧.380
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
9/7/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Female gimbal form blockv2
SIZE
A
DWG. NO.
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License,
Academic Use Only


R.0625
.5000
2.7875
.3750
.6667
.6667
2.4125
2.0000
2X Ø.2010 THRU ALL
Ø.3750 ↧.1900
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
NAME
DATE
DRAWN
J. Giallo
9/8/07
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Female gimbal side form block
SIZE
A
DWG. NO.
REV
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF
SolidWorks Student License,
Academic Use Only

Appendix VI. SUS Survey and Instructions

Laser Control Module Evaluation

This exercise is designed as a trial to evaluate a Laser Control Module. You will be asked to complete three distinct tasks.

TASK 1: Take a few moments to familiarize yourself with the control mechanism for the laser by tracing random patterns. Determine a comfortable arm and hand position prior to beginning the exercise.

TASK 2: On the figure to the left, direct the HeNe beam to each of the numbered spots in numerical order using the joystick controller. Pause at each numbered site and depress the trigger on the joystick followed by firing the laser by activating the foot switch on the CO2 laser for 1 second. Proceed to each numbered site sequentially depressing the trigger once followed by the foot pedal.

After completing Task 2 please wait for instruction from study personnel.

TASK 3: Proceed to the figure on the right. Again, direct the HeNe beam to each of the numbered spots in numerical order using the joystick controller. Pause at each numbered site and depress the trigger on the joystick followed by firing the laser by activating the foot switch on the CO2 laser for 1 second. Proceed to the next numbered site completing the two tasks above at each numbered site.

After completing Task 3 please wait for instruction from study personnel.

Laser Control Module Evaluation

For each of the questions below please circle the most accurate response from **1 (strongly disagre**e) to **5 (strongly agree)**

1. Using the joystick control was natural and intuitive **1 2 3 4 5**

2. The level of force required to manipulate the joystick **1 2 3 4 5**
   was appropriate for the task

3. The use of the trigger control for marking "key points " **1 2 3 4 5**
   was easy to use

4. I thought the system was easy to use **1 2 3 4 5**

5. I thought there was too much inconsistency in this system. **1 2 3 4 5**

6. I found the system very cumbersome to use. **1 2 3 4 5**

How many (micromanipulator) laser operative case have you done? 0-10

10-100

> 100

Comments:

Appendix VII. Master Control Program Software Listing

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

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help AMM

% Last Modified by GUIDE v2.5 03-Dec-2007 14:04:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @AMM_OpeningFcn, ...
                   'gui_OutputFcn',  @AMM_OutputFcn, ...
```

```
                'gui_LayoutFcn',  [] , ...
                'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

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

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


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

% data = guihandles(hObject);
%
% initialize workspace;
% clear;
% Variable declarations
global Mem_path;
Mem_path                    =                    struct('status'                    ,
{'false'},'x_ptr',{1},'y_ptr',{1},'max_x',{0},'max_y',{0},'x_coord',{0},'y_coord',{0});

%declare snake vectors
global XSnake YSnake;                    % contour of the snake
```

```
% create structure of handles
% Configure the joystick in the system
% joysticksetup;
% Create the connection to the joystick
% Define 4 channels: x, y, z and r axis.

%JFG commented out to enable initialization w/o system
      global ai;
      ai=analoginput('joy',3);

% Create the data structure for the four axes and buttons pressed
% 1 - X axis
% 2 - Y axis
% 3 - Z axis
% 4 - R axis
% 5 - buttons pressed (quasy binary data - see below)
% 6 - no of buttons pressed at the time

%JFG commented out to enable initialization
         addchannel(ai,[1 2 3 4 5 6]);
         handles.ai = ai;

% data.ai = ai;
% guidata(hObject,data);
guidata(hObject,handles); % Store the changes.

%Set the joystick sensitivities to  midpoint initially
set(handles.slider10,'Value',0.5);
set(handles.edit10,'String',...
num2str(get(handles.slider10,'Value')));
set(handles.slider12,'Value',0.5);
set(handles.edit11,'String',...
num2str(get(handles.slider12,'Value')));

% Set trace speed to minimum
set(handles.Trace_Speed,'Value',0);
```

```
   set(handles.edit9,'String',...
   num2str(get(handles.Trace_Speed,'Value')));

   %turn off path memorization
   % persistent Memorize_On;
   % Memorize_On = 'false';
   % handles.Memorize_On = Memorize_On;
   % guidata(hObject,handles); % Store the changes.
   %reset the array pointer starting points
   % persistent x_ptr,y_ptr;
   % x_ptr = 0;
   % y_ptr = 0;
   % handles.x_ptr = x_ptr;
   % handles.y_ptr = y_ptr;
   % guidata(hObject,handles); % Store the changes.

   % global alpha beta gamma kappa dmin dmax;     % optimized parameters for the
snake
   sigma=0;
   mu=0.1;
   alpha=0.05;
   beta=15;
   gamma=1;
   kappa=0.1;
   dmin=0.5;
   dmax=2;

   global SerialSetupStatus;
   SerialSetupStatus = 0;
   % global alpha beta gamma kappa dmin dmax;     % parameters for the snake
   % global mu, sigma;

   set(handles.Alpha_parm,'Value',alpha);
   set(handles.Beta_parm,'Value',beta);
   set(handles.Gamma_parm,'Value',gamma);
   set(handles.Kappa_parm,'Value',kappa);
   set(handles.dmin_parm,'Value',dmin);
```

```
set(handles.dmax_parm,'Value',dmax);
set(handles.Mu_parm,'Value',mu);
set(handles.Sigma_parm,'Value',sigma);
set(handles.edit1,'String',...
num2str(get(handles.Alpha_parm,'Value')));
set(handles.edit2,'String',...
num2str(get(handles.Beta_parm,'Value')));
set(handles.edit3,'String',...
num2str(get(handles.Gamma_parm,'Value')));
set(handles.edit4,'String',...
num2str(get(handles.Kappa_parm,'Value')));
set(handles.edit5,'String',...
num2str(get(handles.dmin_parm,'Value')));
set(handles.edit6,'String',...
num2str(get(handles.dmax_parm,'Value')));
set(handles.edit7,'String',...
num2str(get(handles.Mu_parm,'Value')));
set(handles.edit8,'String',...
num2str(get(handles.Sigma_parm,'Value')));

% guidata(hObject,handles); % Store the changes.

%configure the active contour setup
snakesetup;

%JFG comments out to bring up GUI without system attached
AMMSerialSetup;

% AMMSerialSelftest;

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

movegui('center');

% UIWAIT makes AMM wait for user response (see UIRESUME)
```

```
% uiwait(handles.figure1);

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

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

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

menu3('InicSnake');
% menu3('LoadSnake');
menu3('ManualSnakeON');

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

global alpha beta gamma kappa dmin dmax;       % optimized parameters for the snake

sigma=0;
mu=0.1;
alpha=0.05;
beta=15;
gamma=1;
kappa=0.1;
dmin=0.5;
```

```
dmax=2;

menu3('SnakeIter');

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

% plot(get(hObject,'Value',XSnake_parm),get(hObject,'Value',YSnake_parm));

% evalin('base',plot(XSnake,YSnake));
% set(gca,'YDir','reverse');
% plotsnake;
global XSnake YSnake;
global x_snake_max y_snake_max;
global AMMSlave;

% declare variables

plot_x_max = 7075;
plot_x_min = 6350;
% plot_y_max = 6850;
% plot_y_min = 6475;
plot_y_max = 6475;
plot_y_min = 6850;

plot_x_center = (plot_x_max-plot_x_min)/2 + plot_x_min;
plot_y_center = -((plot_y_max-plot_y_min)/2 + plot_y_min);

%Get the current joystick sensitivity
Joystick_X_Sensitivity = get(handles.slider10,'Value');
Joystick_Y_Sensitivity = get(handles.slider12,'Value');

differential_x_max = abs(round(Joystick_X_Sensitivity*(plot_x_max - plot_x_min)));
differential_y_max = abs(round(Joystick_Y_Sensitivity*(plot_y_max - plot_y_min)));
```

```
assignin('base','differential_x_max',differential_x_max);
assignin('base','differential_y_max',differential_y_max);

%Determine the number of points in the snake

xsnake_size = size(XSnake);
% xsnake_size = xsnake_size(1,1);
ysnake_size = size(YSnake);
% ysnake_size = ysnake_size(1,1);

%Make a single matrix with both x & y snake elements

vector_length = length(XSnake);
assignin('base','vector_length',vector_length);
vector_midpoint = round(vector_length/2);

for i=1 : vector_length
   composite_snake(i,1) = (XSnake(i));
   composite_snake(i,2) = (YSnake(i));
end

%find some key metrics in the snake
snake_max = max(composite_snake);
x_snake_max = snake_max(1,1);
y_snake_max = snake_max(1,2);
snake_min = min(composite_snake);
x_snake_min = snake_min(1,1);
y_snake_min = snake_min(1,2);
snake_delta = snake_max - snake_min;
x_snake_center = (snake_delta(1,1)/2) + x_snake_min;
y_snake_center = (snake_delta(1,2)/2) + y_snake_min;
%Scale the GVF snake to the medical robotic system workspace
for i=1:vector_length
      for j=1:2
         if (j == 1)
```

```
            plot_snake(i,j)=round(((composite_snake(i,j)-
x_snake_center)/(snake_delta(1,1)/2))*(differential_x_max/2) + plot_x_center);
         else
            plot_snake(i,j)=round(((composite_snake(i,j)-
y_snake_center)/(snake_delta(1,2)/2))*(differential_y_max/2) + plot_y_center);
         end
      end
   end

   assignin('base','plot_snake',plot_snake);

   %Establish a maximum number of delay tics (in milliseconds)
   Max_Delay = 35;
   %Establish a minimum delay time interval (in milliseconds)
   wait_time = 0.00025;
   % Now plot the snake

   for i = 1:vector_length
      for j=1:2
         if j==1
            %Compute the X coordinate
            JoyXCoordinate = plot_snake(i,j);
            %Output the coordinate to the slave microcontroller
            fprintf(AMMSlave,['X' int2str(JoyXCoordinate)], 'async');
            %To avoid overrunning the serial port we must wait until all five
            %command characters have been sent out
            Port_Occupied = get(AMMSlave,'TransferStatus');
            Delay_iterations = 0;
            while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
               pause(wait_time);
               Port_Occupied = get(AMMSlave,'TransferStatus');
               Delay_iterations = Delay_iterations + 1;
            end
         end
         if j==2
            %Compute the Y coordinate
            JoyYCoordinate = abs(plot_snake(i,j));
```

```
            fprintf(AMMSlave,['Y' int2str(JoyYCoordinate)], 'async');
            %To avoid overrunning the serial port we must wait until all five
            %command characters have been sent out
            Port_Occupied = get(AMMSlave,'TransferStatus');
            Delay_iterations = 0;
            while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
               pause(wait_time);
               Port_Occupied = get(AMMSlave,'TransferStatus');
               Delay_iterations = Delay_iterations + 1;
            end
         end
      end
      pause_time = (1-get(handles.Trace_Speed,'Value'))*1000*wait_time;
      pause(pause_time);

assignin('base','JoyXCoordinate',JoyXCoordinate);
assignin('base','JoyYCoordinate',JoyYCoordinate);
end
assignin('base','AMMSlave',AMMSlave);

%Now ablate within the outline of the tumor
color_snake = sortrows(plot_snake,2);
slow_trace = 0.25;
set(handles.Trace_Speed,'Value',slow_trace);
set(handles.edit9,'String',...
num2str(get(handles.Trace_Speed,'Value')));
%
for i = 1:vector_length
   for j=1:2
      if j==1
         %Compute the X coordinate
         JoyXCoordinate = color_snake(i,j);
         %Output the coordinate to the slave microcontroller
         fprintf(AMMSlave,['X' int2str(JoyXCoordinate)], 'async');
         %To avoid overrunning the serial port we must wait until all five
         %command characters have been sent out
         Port_Occupied = get(AMMSlave,'TransferStatus');
```

```
            Delay_iterations = 0;
            while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
               pause(wait_time);
               Port_Occupied = get(AMMSlave,'TransferStatus');
               Delay_iterations = Delay_iterations + 1;
            end
         end
         if j==2
            %Compute the Y coordinate
            JoyYCoordinate = abs(color_snake(i,j));
            fprintf(AMMSlave,['Y' int2str(JoyYCoordinate)], 'async');
            %To avoid overrunning the serial port we must wait until all five
            %command characters have been sent out
            Port_Occupied = get(AMMSlave,'TransferStatus');
            Delay_iterations = 0;
            while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
               pause(wait_time);
               Port_Occupied = get(AMMSlave,'TransferStatus');
               Delay_iterations = Delay_iterations + 1;
            end
         end
      end
      pause_time = (1-get(handles.Trace_Speed,'Value'))*1000*wait_time;
      pause(pause_time);

assignin('base','JoyXCoordinate',JoyXCoordinate);
assignin('base','JoyYCoordinate',JoyYCoordinate);
end
assignin('base','color_snake',color_snake);
assignin('base','AMMSlave',AMMSlave);
%recenter HeNe beam

% % pause(0.25);
% fprintf(AMMSlave,'X6713', 'async');
% % pause(0.25);
% Port_Occupied = get(AMMSlave,'TransferStatus');
%           Delay_iterations = 0;
```

```
%           while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
%              pause(wait_time);
%              Port_Occupied = get(AMMSlave,'TransferStatus');
%              Delay_iterations = Delay_iterations + 1;
%           end
% fprintf(AMMSlave,'Y6663', 'async');
% % pause(0.25);
% Port_Occupied = get(AMMSlave,'TransferStatus');
%           Delay_iterations = 0;
%           while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
%              pause(wait_time);
%              Port_Occupied = get(AMMSlave,'TransferStatus');
%              Delay_iterations = Delay_iterations + 1;
%           end
%
% plot_x = [plot_snake(:,1)];
% plot_y = [plot_snake(:,2)];
% figure;
% plot(plot_x,plot_y);

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

%and do a self test of the servos.

AMMSerialSelftest;

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

% Hints: contents = get(hObject,'String') returns ParmSel contents as cell array
```

```
%       contents{get(hObject,'Value')} returns selected item from ParmSel

% % Determine the selected data set.
% str = get(hObject, 'String');
val = get(hObject,'Value');
% Set current data to the selected data set.

switch val;
case 1 % Use optimized parms.
%
global alpha beta gamma kappa dmin dmax;      % parameters for the snake
sigma=0;
mu=0.1;
alpha=0.05;
beta=15;
gamma=1;
kappa=0.1;
dmin=0.5;
dmax=2;
set(handles.Alpha_parm,'Value',alpha);
set(handles.Beta_parm,'Value',beta);
set(handles.Gamma_parm,'Value',gamma);
set(handles.Kappa_parm,'Value',kappa);
set(handles.dmin_parm,'Value',dmin);
set(handles.dmax_parm,'Value',dmax);
set(handles.Mu_parm,'Value',mu);
set(handles.Sigma_parm,'Value',sigma);
set(handles.edit1,'String',...
num2str(get(handles.Alpha_parm,'Value')));
set(handles.edit2,'String',...
num2str(get(handles.Beta_parm,'Value')));
set(handles.edit3,'String',...
num2str(get(handles.Gamma_parm,'Value')));
set(handles.edit4,'String',...
num2str(get(handles.Kappa_parm,'Value')));
set(handles.edit5,'String',...
num2str(get(handles.dmin_parm,'Value')));
```

```
set(handles.edit6,'String',...
num2str(get(handles.dmax_parm,'Value')));
set(handles.edit7,'String',...
num2str(get(handles.Mu_parm,'Value')));
set(handles.edit8,'String',...
num2str(get(handles.Sigma_parm,'Value')));

case 2 % Use default parms.
global alpha beta gamma kappa dmin dmax;      % parameters for the snake
sigma=0;
mu=0.1;
alpha=0.05;
beta=0;
gamma=1;
kappa=0.6;
dmin=0.5;
dmax=2;

set(handles.Alpha_parm,'Value',alpha);
set(handles.Beta_parm,'Value',beta);
set(handles.Gamma_parm,'Value',gamma);
set(handles.Kappa_parm,'Value',kappa);
set(handles.dmin_parm,'Value',dmin);
set(handles.dmax_parm,'Value',dmax);
set(handles.Mu_parm,'Value',mu);
set(handles.Sigma_parm,'Value',sigma);
set(handles.edit1,'String',...
   num2str(get(handles.Alpha_parm,'Value')));
set(handles.edit2,'String',...
   num2str(get(handles.Beta_parm,'Value')));
set(handles.edit3,'String',...
   num2str(get(handles.Gamma_parm,'Value')));
set(handles.edit4,'String',...
   num2str(get(handles.Kappa_parm,'Value')));
set(handles.edit5,'String',...
   num2str(get(handles.dmin_parm,'Value')));
set(handles.edit6,'String',...
```

```
    num2str(get(handles.dmax_parm,'Value')));
set(handles.edit7,'String',...
    num2str(get(handles.Mu_parm,'Value')));
set(handles.edit8,'String',...
    num2str(get(handles.Sigma_parm,'Value')));
case 3 % load custom parms from file
    global alpha beta gamma kappa dmin dmax;      % parameters for the snake
    menu3('LoadSnake');
    set(handles.Alpha_parm,'Value',alpha);
    set(handles.Beta_parm,'Value',beta);
    set(handles.Gamma_parm,'Value',gamma);
    set(handles.Kappa_parm,'Value',kappa);
    set(handles.dmin_parm,'Value',dmin);
    set(handles.dmax_parm,'Value',dmax);
    set(handles.Mu_parm,'Value',mu);
    set(handles.Sigma_parm,'Value',sigma);
    set(handles.edit1,'String',...
        num2str(get(handles.Alpha_parm,'Value')));
    set(handles.edit2,'String',...
        num2str(get(handles.Beta_parm,'Value')));
    set(handles.edit3,'String',...
        num2str(get(handles.Gamma_parm,'Value')));
    set(handles.edit4,'String',...
        num2str(get(handles.Kappa_parm,'Value')));
    set(handles.edit5,'String',...
        num2str(get(handles.dmin_parm,'Value')));
    set(handles.edit6,'String',...
        num2str(get(handles.dmax_parm,'Value')));
    set(handles.edit7,'String',...
        num2str(get(handles.Mu_parm,'Value')));
    set(handles.edit8,'String',...
        num2str(get(handles.Sigma_parm,'Value')));
case 4 % save custom parms to file
    global alpha beta gamma kappa dmin dmax;      % parameters for the snake
    menu3('SaveSnake');
end
% Save the handles structure.
```

```
guidata(hObject,handles);

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

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

% --- Executes on slider movement.
function Alpha_parm_Callback(hObject, eventdata, handles)
% hObject    handle to Alpha_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit1,'String',...
num2str(get(handles.Alpha_parm,'Value')));
guidata(hObject,handles); % Store the changes.
% --- Executes during object creation, after setting all properties.
function Alpha_parm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Alpha_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
```

```
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


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

% get the opening image
global CallbackOpen;
menu1('CallbackOpen');
menu2;
%% JFG inserts this to force the field calculation
caclVF;

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




function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double
val = str2double(get(hObject,'String'));
```

```
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Alpha_parm,'Min') && ...
val <= get(handles.Alpha_parm,'Max')
set(handles.Alpha_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.
% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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




% --- Executes on slider movement.
function Beta_parm_Callback(hObject, eventdata, handles)
% hObject    handle to Beta_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit2,'String',...
num2str(get(handles.Beta_parm,'Value')));
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
```

```
function Beta_parm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Beta_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function Gamma_parm_Callback(hObject, eventdata, handles)
% hObject    handle to Gamma_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit3,'String',...
num2str(get(handles.Gamma_parm,'Value')));
guidata(hObject,handles); % Store the changes.

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

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
```

```
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function Kappa_parm_Callback(hObject, eventdata, handles)
% hObject    handle to Kappa_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit4,'String',...
num2str(get(handles.Kappa_parm,'Value')));
guidata(hObject,handles); % Store the changes.

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

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function dmin_parm_Callback(hObject, eventdata, handles)
% hObject    handle to dmin_parm (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit5,'String',...
num2str(get(handles.dmin_parm,'Value')));
guidata(hObject,handles); % Store the changes.

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

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function dmax_parm_Callback(hObject, eventdata, handles)
% hObject    handle to dmax_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit6,'String',...
num2str(get(handles.dmax_parm,'Value')));
guidata(hObject,handles); % Store the changes.
```

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

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function Mu_parm_Callback(hObject, eventdata, handles)
% hObject    handle to Mu_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit7,'String',...
num2str(get(handles.Mu_parm,'Value')));
guidata(hObject,handles); % Store the changes.

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

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
```

```
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function Sigma_parm_Callback(hObject, eventdata, handles)
% hObject    handle to Sigma_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit8,'String',...
num2str(get(handles.Sigma_parm,'Value')));
guidata(hObject,handles); % Store the changes.
% --- Executes during object creation, after setting all properties.
function Sigma_parm_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Sigma_parm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on slider movement.
function Trace_Speed_Callback(hObject, eventdata, handles)
% hObject    handle to Trace_Speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

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

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit9,'String',...
num2str(get(handles.Trace_Speed,'Value')));
guidata(hObject,handles); % Store the changes.

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

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function slider10_Callback(hObject, eventdata, handles)
% hObject    handle to slider10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit10,'String',...
num2str(get(handles.slider10,'Value')));
guidata(hObject,handles); % Store the changes.
```

```
% --- Executes during object creation, after setting all properties.
function slider10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Beta_parm,'Min') && ...
val <= get(handles.Beta_parm,'Max')
set(handles.Beta_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Gamma_parm,'Min') && ...
val <= get(handles.Gamma_parm,'Max')
set(handles.Gamma_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Kappa_parm,'Min') && ...
val <= get(handles.Kappa_parm,'Max')
set(handles.Kappa_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

```
function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.dmin_parm,'Min') && ...
val <= get(handles.dmin_parm,'Max')
set(handles.dmin_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.dmax_parm,'Min') && ...
val <= get(handles.dmax_parm,'Max')
set(handles.dmax_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a double
```

```
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Mu_parm,'Min') && ...
val <= get(handles.Mu_parm,'Max')
set(handles.Mu_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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



function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Sigma_parm,'Min') && ...
val <= get(handles.Sigma_parm,'Max')
```

```
set(handles.Sigma_parm,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

function edit9_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.Trace_Speed,'Min') && ...
val <= get(handles.Trace_Speed,'Max')
set(handles.Trace_Speed,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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


function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.slider10,'Min') && ...
val <= get(handles.slider10,'Max')
set(handles.slider10,'Value',val);
end;
guidata(hObject,handles); % Store the changes.


% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%JFG commented out to enable use w/o system connected

%clean up serial port use
global AMMSlave;
 if(strcmp(class(AMMSlave),'double'))
     fclose(AMMSlave);
 end

% Hint: delete(hObject) closes the figure
delete(hObject);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                        %
%              Activate Joystick function                    %
%                                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% --- Executes on button press in Enable_Joystick.
function Enable_Joystick_Callback(hObject, eventdata, handles)
% hObject    handle to Enable_Joystick (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of Enable_Joystick
global UpdateJoy;
global Mem_path;
button_state = get(hObject,'Value');
if button_state == get(hObject,'Max')
   % toggle button is pressed
   UpdateJoy              =              timer('TimerFcn',{@UpdateJoy_Callback,handles},
'ExecutionMode',...
      'fixedRate','Period', 0.08,'BusyMode','queue');
   start(UpdateJoy);
   handles.UpdateJoy = UpdateJoy;
   set(gcbo,'String','Disable Joystick');
%    ,...
%    'CallBack',{@JoystickOff_Callback,handles});
   guidata(hObject,handles); % Store the changes.
elseif button_state == get(hObject,'Min')
   % toggle button is not pressed

   set(gcbo,'String','Enable Joystick');
   stop(UpdateJoy);
   handles.UpdateJoy = UpdateJoy;
   % delete(UpdateJoy);
   guidata(hObject,handles); % Store the changes.
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                              %
%          Activate Disable Joystick function           %
%                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%this routine eliminates the timer object and resets the button text
function JoystickOff_Callback(hObject,event,handles)
global UpdateJoy;
set(gcbo,'String','Enable Joystick');
stop(UpdateJoy);
% delete(UpdateJoy);
guidata(hObject,handles); % Store the changes.
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                    %
%              Activate Playback path function                   %
%                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% --- Executes on button press in Playback.
function Playback_Callback(hObject, eventdata, handles)
% hObject    handle to Playback (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global Mem_path;
global UpdateJoy;
global AMMSlave;
%Establish a maximum number of delay tics (in milliseconds)
Max_Delay = 50;
%Establish a minimum delay time interval (in milliseconds)
wait_time = 0.0005;
%Turn off the joystick if it is on
Joystick_state = get(UpdateJoy, 'Running');
switch Joystick_state
   case {'on'}
      %suspend the joystick operation during playback
      stop(UpdateJoy);
      handles.UpdateJoy = UpdateJoy;
```

```
        guidata(hObject,handles); % Store the changes.
        %set a flag that we turned off the joystick
        playbackhalt = 'true';
    case {'off'}
        playbackhalt = 'false';
end
% delete(UpdateJoy);
guidata(hObject,handles); % Store the changes.
%set up the playback
Mem_path.x_ptr = 1;
Mem_path.y_ptr = 1;
while Mem_path.x_ptr <= Mem_path.max_x
        Port_Occupied = get(AMMSlave,'TransferStatus');
        Delay_iterations = 0;
        while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
            pause(wait_time);
            Port_Occupied = get(AMMSlave,'TransferStatus');
            Delay_iterations = Delay_iterations + 1;
        end
        fprintf(AMMSlave,['X' int2str(Mem_path.x_coord(Mem_path.x_ptr))], 'async');
        %To avoid overrunning the serial port we must wait until all five
        %command characters have been sent out
        Port_Occupied = get(AMMSlave,'TransferStatus');
        Delay_iterations = 0;
        while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
            pause(wait_time);
            Port_Occupied = get(AMMSlave,'TransferStatus');
            Delay_iterations = Delay_iterations + 1;
        end
        fprintf(AMMSlave,['Y' int2str(Mem_path.y_coord(Mem_path.y_ptr))], 'async');
        %To avoid overrunning the serial port we must wait until all five
        %command characters have been sent out
        Port_Occupied = get(AMMSlave,'TransferStatus');
        Delay_iterations = 0;
        while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
            pause(wait_time);
            Port_Occupied = get(AMMSlave,'TransferStatus');
```

```
            Delay_iterations = Delay_iterations + 1;
        end
        Mem_path.x_ptr = Mem_path.x_ptr + 1;
        Mem_path.y_ptr = Mem_path.y_ptr + 1;
        pause_time = (1-get(handles.Trace_Speed,'Value'))*1000*wait_time;
        pause(pause_time);
end
% Joystick_state = get(UpdateJoy, 'Running');
% switch Joystick_state
%     case {'off'}
%         %re-enable joystick after playback
%         start(UpdateJoy);
%         Delay_iterations = 0;
%         Long_Max_Delay = Max_Delay * 2;
%         pause(wait_time * Max_Delay);
%         while (strcmp(Joystick_state,'off') & (Delay_iterations <= Long_Max_Delay))
%             pause(wait_time);
%             Joystick_state = get(UpdateJoy, 'Running');
%             Delay_iterations = Delay_iterations + 1;
%         end
% %         handles.UpdateJoy = UpdateJoy;
% %         guidata(hObject,handles); % Store the changes.
%         %set a flag that we turned off the joystick
%     case {'on'}
%         %no action required
% end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                               %
%             Activate  Memorize path function                %
%                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% --- Executes on button press in Memorize_path.
```

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

% global Memorize_On;
% Memorize_On = 'setup';
% handles.Memorize_On = Memorize_On;
% guidata(hObject,handles); % Store the changes.
% handles.x_ptr = x_ptr;
% guidata(hObject,handles); % Store the changes.
% handles.y_ptr = y_ptr;
% guidata(hObject,handles); % Store the changes.
global Mem_path;
global UpdateJoy;
button_state = get(hObject,'Value');
if button_state == get(hObject,'Max')
   % toggle button is pressed
   % Make sure the joystick is on
   Joystick_state = get(UpdateJoy, 'Running');
   switch Joystick_state
      case {'off'}
         %suspend the joystick operation during playback
         start(UpdateJoy);
         handles.UpdateJoy = UpdateJoy;
         guidata(hObject,handles); % Store the changes.
      case {'off'}
         %do nothing
   end
   Mem_path.x_ptr = 1;
   Mem_path.y_ptr = 1;
   %Clear out the stored path arrays
   while Mem_path.x_ptr <= Mem_path.max_x
      Mem_path.x_coord(Mem_path.x_ptr) = 0;
      Mem_path.y_coord(Mem_path.y_ptr) = 0;
      Mem_path.x_ptr = Mem_path.x_ptr + 1;
      Mem_path.y_ptr = Mem_path.y_ptr + 1;
```

```
    end
    %Initialize array pointers to capture new path
    Mem_path.x_ptr = 1;
    Mem_path.y_ptr = 1;
    Mem_path.max_x = 0;
    Mem_path.max_y = 0;
    Mem_path.status = 'true';
    set(handles.Memorize_path,'String','Memorize Off');
%     ,...
%         'CallBack',{@Memorize_Off_Callback,handles});
    guidata(hObject,handles); % Store the changes.
elseif button_state == get(hObject,'Min')
    % toggle button is not pressed
    Mem_path.max_x = Mem_path.x_ptr - 1;
    Mem_path.max_y = Mem_path.y_ptr - 1;
    %reset the array pointer starting points
    Mem_path.x_ptr = 1;
    Mem_path.y_ptr = 1;
    Mem_path.status = 'false';
    set(gcbo,'String','Memorize Path');
    guidata(hObject,handles); % Store the changes.
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        %
%            Update Joystick                    %
%                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function UpdateJoy_Callback(hObject,eventdata,handles)
% tic
global AMMSlave;
% global x_ptr;
% global y_ptr;
global Mem_path;
% handles = guidata(handles.figure1);
```

```
  % global x_ptr,y_ptr;
  % Set the range of motion for the mechatronics
  JoyXMax = 7075;
  JoyXMin = 6350;
  JoyYMax = 6850;
  JoyYMin = 6475;
  %Establish a maximum number of delay tics (in milliseconds)
  Max_Delay = 35;
  %Establish a minimum delay time interval (in milliseconds)
  wait_time = 0.00025;
  %Get the current joystick position
  d = getsample(handles.ai);
  %Get the current joystick sensitivity
  Joystick_X_Sensitivity = get(handles.slider10,'Value');
  Joystick_Y_Sensitivity = get(handles.slider12,'Value');
    %Determine where the joystick is being moved and export an appropriate
    %motion command to the slave microcontroller
    if d(1)<0
      %the joystick is deflecting to the left on the x-axis
      %and the range of slave values is from 6350 to 6712
      %Compute the X coordinate
      JoyXCoordinate = round((d(1)/10)*Joystick_X_Sensitivity*((JoyXMax-
JoyXMin)/2)+(JoyXMin+(JoyXMax-JoyXMin)/2));
      %Output the coordinate to the slave microcontroller
      fprintf(AMMSlave,['X' int2str(JoyXCoordinate)], 'async');
      %To avoid overrunning the serial port we must wait until all five
      %command characters have been sent out
      Port_Occupied = get(AMMSlave,'TransferStatus');
      Delay_iterations = 0;
      while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
        pause(wait_time);
        Port_Occupied = get(AMMSlave,'TransferStatus');
        Delay_iterations = Delay_iterations + 1;
      end
    end
    if d(1)>0
```

```
        %the joystick is deflecting to the right on the x-axis
        %and the range of slave values is from 6712 to 7075
        %Compute the X coordinate
        JoyXCoordinate = round((d(1)/10)*Joystick_X_Sensitivity*((JoyXMax-
JoyXMin)/2)+(JoyXMin+(JoyXMax-JoyXMin)/2));
        %Output the coordinate to the slave microcontroller
        fprintf(AMMSlave,['X' int2str(JoyXCoordinate)], 'async');
        %To avoid overrunning the serial port we must wait until all five
        %command characters have been sent out
        Port_Occupied = get(AMMSlave,'TransferStatus');
        Delay_iterations = 0;
        while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
           pause(wait_time);
           Port_Occupied = get(AMMSlave,'TransferStatus');
           Delay_iterations = Delay_iterations + 1;
        end

     end
     if d(2)<0
        %the joystick is deflecting down on the y-axis
        %and the range of slave values is from 6475 to 6650
        %Compute the Y coordinate
        JoyYCoordinate = round((-d(2)/10)*Joystick_Y_Sensitivity*((JoyYMax-
JoyYMin)/2)+(JoyYMin+(JoyYMax-JoyYMin)/2));
        %Output the coordinate to the slave microcontroller
        fprintf(AMMSlave,['Y' int2str(JoyYCoordinate)], 'async');
        %To avoid overrunning the serial port we must wait until all five
        %command characters have been sent out
        Port_Occupied = get(AMMSlave,'TransferStatus');
        Delay_iterations = 0;
        while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
           pause(wait_time);
           Port_Occupied = get(AMMSlave,'TransferStatus');
           Delay_iterations = Delay_iterations + 1;
        end
     end
     if d(2)>0
```

```
        %the joystick is deflecting up on the y-axis
         %and the range of slave values is from 6650 to 6850
         %Compute the Y coordinate
         JoyYCoordinate = round((-d(2)/10)*Joystick_Y_Sensitivity*((JoyYMax-
JoyYMin)/2)+(JoyYMin+(JoyYMax-JoyYMin)/2));
         %Output the coordinate to the slave microcontroller
         fprintf(AMMSlave,['Y' int2str(JoyYCoordinate)], 'async');
         %To avoid overrunning the serial port we must wait until all five
         %command characters have been sent out
         Port_Occupied = get(AMMSlave,'TransferStatus');
         Delay_iterations = 0;
         while (strcmp(Port_Occupied,'write') & (Delay_iterations <= Max_Delay))
            pause(wait_time);
            Port_Occupied = get(AMMSlave,'TransferStatus');
            Delay_iterations = Delay_iterations + 1;
         end
      end
   %   Now determine if Memorize data points is enabled and if so,
   %   if any buttons were pressed and, if so, if the trigger was one of them
      switch Mem_path.status
         case {'setup'}
   %           global x_ptr,y_ptr;
   %           x_ptr = 0;
   %           y_ptr = 0;
            Mem_path.status = 'true';
            Mem_path.x_ptr = 1;
            Mem_path.y_ptr = 1;
         case {'true'}
          ButtonsPressed = round((10+d(5))*32768/10);
          bP = ButtonsPressed;
         switch bP;
         % %   If bP = 'trigger pressed'
         %     then start creating the stored data points
            case 1
               Mem_path.x_coord(Mem_path.x_ptr) = JoyXCoordinate;
               Mem_path.y_coord(Mem_path.y_ptr) = JoyYCoordinate;
               Mem_path.x_ptr = Mem_path.x_ptr + 1;
```

```
            Mem_path.y_ptr = Mem_path.y_ptr + 1;
            bP = 0; %debounce the indicator to only record once per trigger pull
        end
        case {'false'}
          % no action required
    end
% toc
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                   %
%           Turn off path memorization              %
%                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Memorize_Off_Callback(hObject,event,handles)
global Mem_path;
% global x_ptr;
% global y_ptr;
% max_xptr,max_yptr;
%reset the button text

%capture the maximum pointer values for playback
Mem_path.max_x = Mem_path.x_ptr - 1;
Mem_path.max_y = Mem_path.y_ptr - 1;
%reset the array pointer starting points
Mem_path.x_ptr = 1;
Mem_path.y_ptr = 1;
Mem_path.status = 'false';
% x_ptr = 0;
% y_ptr = 0;
set(handles.Memorize_path,'String','Memorize path');
guidata(hObject,handles); % Store the changes.
return
```

```
% --- Executes on slider movement.
function slider12_Callback(hObject, eventdata, handles)
% hObject    handle to slider12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.edit11,'String',...
num2str(get(handles.slider12,'Value')));
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function slider12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background, change
%       'usewhitebg' to 0 to use default.  See ISPC and COMPUTER.
usewhitebg = 1;
if usewhitebg
   set(hObject,'BackgroundColor',[.9 .9 .9]);
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end




function edit11_Callback(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
```

```
%        str2double(get(hObject,'String')) returns contents of edit11 as a double
val = str2double(get(hObject,'String'));
% Determine whether val is a number between 0 and 1.
if isnumeric(val) && length(val)==1 && ...
val >= get(handles.slider12,'Min') && ...
val <= get(handles.slider12,'Max')
set(handles.slider12,'Value',val);
end;
guidata(hObject,handles); % Store the changes.

% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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




% --------------------------------------------------------------------
function uipanel3_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% global UpdateJoy;
% global Mem_path;
% selection = get(hObject,'SelectedObject');
% switch get(selection,'Tag')
%     case 'Enable_Joystick'
```

```
%     button_state = get(hObject,'Value');
%     if button_state == get(hObject,'Max')
%        % toggle button is pressed
%                        UpdateJoy   =   timer('TimerFcn',{@UpdateJoy_Callback,handles},
'ExecutionMode','fixedRate','Period', 0.08);
%        start(UpdateJoy);
%        handles.UpdateJoy = UpdateJoy;
%        set(gcbo,'String','Disable Joystick');
%        %     ,...
%        %    'CallBack',{@JoystickOff_Callback,handles});
%        guidata(hObject,handles); % Store the changes.
%     elseif button_state == get(hObject,'Min')
%        % toggle button is not pressed
%
%        set(gcbo,'String','Enable Joystick');
%        stop(UpdateJoy);
%        % delete(UpdateJoy);
%        guidata(hObject,handles); % Store the changes.
%     end
%     case 'radiobutton2'
       % code piece when radiobutton2 is selected goes here
       % ...
% end
% This routine initializes the AMM serial port
% and enables outgoing command communication
% from the master AMM controller to the slave
%
function AMMSerialSetup()

global AMMSlave;
global SerialSetupStatus;

if SerialSetupStatus == 0
   if(strcmp(class(AMMSlave),'double'))
      AMMSlave = serial('COM5');
      set(AMMSlave,'BaudRate',460800,  'DataBits',  8,  'StopBits',  1,  'Parity','none',
'Terminator', 13);
```

```
    end
    pause(0.1);
    SerialSetupStatus = 1;
    PortStatus = (get(AMMSlave,'status'));
    if (strcmp(PortStatus,'closed'))
        fopen(AMMSlave);
        pause(0.1);
    end
end
return
```

Appendix VIII. Slave Control Program Software Listing

```
//-------------------------------------------------------------------
---------
//
// Automated Surgical laser micromanipulator controller
//
// This control program implements messaging and PWM interfaces
// for the purpose of controlling two servos that in turn operate a
manual
// micromanipulator that guides a surgical laser.  The life support
functions
// of the MSP430 and the serial USB interface are performed by the demo
code
// described in the following paragraph.
//
// Joseph F. Giallo, II
// Joint Department of Biomedical Engineering
// N.C. State University and the University of North Carolina at Chapel
Hill
// April 2008
// Version 2.0
//
//
// MSP430-TUSB3410 Reference Design Demo Code V1.00
//
// This demo software interfaces the MSP430 to the TUSB3410 USB-to-
serial
// bridge controller. The program is part of the "MSP430 USB
Connectivity
// using TUSB3410" application note (SLAA276). It is demonstrated how a
blank
// USB configuration EEPROM is detected, and how it gets programmed
with an
// image that is stored in MSP430 Flash memory. After this, a demo
application
// is started. The demo receives characters from the TUSB3410 virtual
COM port
// (VCP) with a baud rate of 460,800. The lower nibble of a received
character
// is output to the LEDs 1...4 on the demo board. Also, on press or
release of
// any push-button SW1...SW4, a byte is transmitted back to the PC
containing
// the updated button state.
//
// MSP430 ressources used:
//   o USART0 in UART mode for serial comm with TUSB3410
```

```
//   o USART0 in I2C mode for comm with EEPROM
//   o Timer_B7 for push-button query
//
// Andreas Dannenberg
// MSP430/TMS470 Applications
// Texas Instruments Inc.
// November 2005
//
// Built with IAR Embedded Workbench V3.30
//------------------------------------------------------------------------
---------
#include "msp430x16x.h"
#include "ctype.h"
#include <stdlib.h>
#include <string.h>
//------------------------------------------------------------------------
---------
// I2C communication related definitions
//------------------------------------------------------------------------
---------
#define EEPROM_ADDRESS          0x50          // Address of EEPROM
#define OWN_ADDRESS             0x40          // I2C own address
//------------------------------------------------------------------------
---------
// Operate TUSB3410 RESET signal as open-drain output
//------------------------------------------------------------------------
---------
#define TUSB3410_RESET_LOW      P3DIR |= 0x01
#define TUSB3410_RESET_HIGHZ    P3DIR &= ~0x01
//------------------------------------------------------------------------
---------
// Misc definitions
//------------------------------------------------------------------------
---------
enum { false, true };
//------------------------------------------------------------------------
---------
// Globals used by I2C routines
//------------------------------------------------------------------------
---------
static unsigned char I2CBuffer[3];
static unsigned char I2CTxPtr;
//------------------------------------------------------------------------
---------
// Globals used by main() and ISRs
//------------------------------------------------------------------------
---------
static unsigned char ButtonState = 0;
```

```
static unsigned char ButtonSet = 0;
static unsigned char ButtonReleased = 0;
static unsigned char RxData;
static unsigned char DataReceived = 0;
static unsigned char x_received = 0;
static unsigned char y_received = 0;
static unsigned char x_char[4];
static unsigned char y_char[4];
static unsigned char x_count = 0;
static unsigned char y_count = 0;
static unsigned int x_coordinate = 0;
static unsigned int y_coordinate = 0;
static unsigned char x_range_ok = false;
static unsigned char y_range_ok = false;
//-------------------------------------------------------------------------
---------
// The constant EEPROMImage[] contains the data to be loaded into the
TUSB3410
// USB configuration EEPROM. It was generated using the tools provided
in
// SLLC251 and then converted into a C constant. Note that the USB
descriptor
// blocks contain checksums, therefore manual modification of the below
EEPROM
// image is not recommended.
//
// USB vendor ID: 0x0451 (TI's VID)
// USB product ID: 0xbeef (This application's PID)
// USB product descriptor: "MSP430-TUSB3410 Reference Design"
//-------------------------------------------------------------------------
---------
static const unsigned char EEPROMImage[] =
{
  0x10, 0x34, 0x03, 0x12, 0x00, 0x33, 0x12, 0x01,
  0x10, 0x01, 0xff, 0x00, 0x00, 0x08, 0x51, 0x04,
  0xef, 0xbe, 0x01, 0x01, 0x01, 0x02, 0x00, 0x01,
  0x05, 0x6c, 0x00, 0x34, 0x04, 0x03, 0x09, 0x04,
  0x24, 0x03, 0x54, 0x00, 0x65, 0x00, 0x78, 0x00,
  0x61, 0x00, 0x73, 0x00, 0x20, 0x00, 0x49, 0x00,
  0x6e, 0x00, 0x73, 0x00, 0x74, 0x00, 0x72, 0x00,
  0x75, 0x00, 0x6d, 0x00, 0x65, 0x00, 0x6e, 0x00,
  0x74, 0x00, 0x73, 0x00, 0x42, 0x03, 0x4d, 0x00,
  0x53, 0x00, 0x50, 0x00, 0x34, 0x00, 0x33, 0x00,
  0x30, 0x00, 0x2d, 0x00, 0x54, 0x00, 0x55, 0x00,
  0x53, 0x00, 0x42, 0x00, 0x33, 0x00, 0x34, 0x00,
  0x31, 0x00, 0x30, 0x00, 0x20, 0x00, 0x52, 0x00,
  0x65, 0x00, 0x66, 0x00, 0x65, 0x00, 0x72, 0x00,
  0x65, 0x00, 0x6e, 0x00, 0x63, 0x00, 0x65, 0x00,
```

```
  0x20, 0x00, 0x44, 0x00, 0x65, 0x00, 0x73, 0x00,
  0x69, 0x00, 0x67, 0x00, 0x6e, 0x00, 0x00, 0x00,
  0x00
};
//-------------------------------------------------------------------------
---------
// Function prototypes
//-------------------------------------------------------------------------
---------
void InitSystem(void);
void InitUART(void);
void Delay100(void);
void InitI2C(void);
void range_check(void);
void update_servo_position(void);
void convert_coordinates(void);
unsigned char EEPROM_CheckNACK(void);
void EEPROM_ByteWrite(unsigned int Address, unsigned char Data);
unsigned char EEPROM_CurrentAddressRead(void);
unsigned char EEPROM_RandomRead(unsigned int Address);
void EEPROM_Write(unsigned int Address, unsigned char *Buffer,
                  unsigned int Count);
void EEPROM_Read(unsigned int Address, unsigned char *Buffer,
                 unsigned int Count);
unsigned int EEPROM_Verify(unsigned int Address, unsigned char *Buffer,
                           unsigned int Count);
//-------------------------------------------------------------------------
---------
// void main(void)
//
// The main application code first initializes the MSP430 peripherals.
Then,
// an event-processing loop is entered. While no event is pending, the
MSP430
// rests in LPM0 with interrupts enabled.
//
// IN:  ButtonState
//      ButtonSet
//      ButtonReleased
//      DataReceived
//      RxData
// OUT: ButtonState
//      ButtonSet
//      ButtonReleased
//      DataReceived
//-------------------------------------------------------------------------
---------
void main(void)
```

```
{
  InitSystem();

  // Initializes Timer_B7 to query the status of the push-buttons
  // SW1, SW2, SW3, and SW4
  TBCCTL0 = CM_1 + SCS + CAP + CCIE;            // Capt. rising edge of
P4.0
  TBCCTL2 = CM_3 + SCS + CAP + CCIE;            // Capt. rising edge of
P4.2
  TBCCTL4 = CM_3 + SCS + CAP + CCIE;            // Capt. rising edge of
P4.4
  TBCCTL6 = CM_3 + SCS + CAP + CCIE;            // Capt. rising edge of
P4.6
  TBCTL = TBSSEL_1 + ID_3 + MC_2;               // ACLK / 8, continuous
mode

  // Activate UART0 RX interrupt operation
  IE1 |= URXIE0;                                // Enable USART0 RX
interrupt
  IFG1 &= ~URXIFG0;                             // Ensure flag is
cleared

  while (1)                                     // Main event
processing loop
  {
    __disable_interrupt();                      // Protect the
following code
    if (!ButtonSet && !ButtonReleased && !DataReceived)
    {
      __bis_SR_register(LPM0_bits + GIE);       // Wait in LPM0 for
event, int
      __no_operation();
    }
    else
      __enable_interrupt();                     // Allow interrupts
again

    if (ButtonSet || ButtonReleased)            // Button event
pending?
    {
      while (!(IFG1 & UTXIFG0));                // Ensure USART is
ready
      __disable_interrupt();                    // Protect the
following code
      U0TXBUF = ButtonState;                    // TX current button
state
      ButtonSet = 0;                            // Clear flags
      ButtonReleased = 0;
```

```
      __enable_interrupt();                     // Allow interrupts
again
    }

    if (DataReceived)                           // UART RX revent
pending?
    {
      __disable_interrupt();                    // Protect the
following code
                                                //Determine what was
received:

//            if (RxData & 0x01)
//                                                  // Check bit 0 of
RX'd char...
//                P4OUT |= 0x02;                        // ...and
set LED1 accordingly
//              else
//                P4OUT &= ~0x02;
//
//              if (RxData & 0x02)                    // Check bit
1 of RX'd char...
//                P4OUT |= 0x08;                       // ...and
set LED2 accordingly
//              else
//                P4OUT &= ~0x08;
//
//              if (RxData & 0x04)                    // Check bit
2 of RX'd char...
//                P4OUT |= 0x20;                       // ...and
set LED3 accordingly
//              else
//                P4OUT &= ~0x20;
//
//              if (RxData & 0x08)                    // Check bit
3 of RX'd char...
//                P4OUT |= 0x80;                       // ...and
set LED4 accordingly
//              else
//                P4OUT &= ~0x80;
//     P4OUT = 0x00;                             // Clear output
latch - turn off all LED's

     if (RxData == 'X' )                        //this is the first
char of a valid command
     {
       x_received = true;
       //flag message start with LED 1
```

```
       P4OUT |= 0x02;                          // ...and set LED1
accordingly
     }
      if (RxData == 'Y' )                             //this is the
first char of a valid command
     {
       y_received = true;
        //flag message start with LED 1
       P4OUT |= 0x02;                          // ...and set LED1
accordingly
     }
     if (isdigit(RxData) && (x_received == true))
// this is the second char of a valid command
       {
          x_char[x_count] =  (RxData & 0x0F);
          x_count++;
         }

     else
      if (isdigit(RxData) && y_received == true)
         {
          y_char[y_count] = (RxData & 0x0F);
          y_count++;
         }

     if (RxData == 0x0d)
     {
       //this is the end of the message
       //toggle LED1 off to flag message received
       P4OUT &= ~0x02;                           // ...and set LED1
accordingly
       //set LED2 on to time how long the processing takes
       P4OUT |= 0x08;                          // ...and set LED2
accordingly
       // reset count vars
       x_count = 0;
       y_count = 0;
       // reset char received vars
       x_received = false;
       y_received = false;
       //reset range checks
        x_range_ok = false;
        y_range_ok = false;
       // call char conversion routine
       convert_coordinates();
       // check to see if the coordinates are valid
       range_check();
       // call update servo position
```

```
        update_servo_position();
        // turn LED 2 off now
        P4OUT &= ~0x08;
      }
       DataReceived = false;                    // Event was handled
       __enable_interrupt();                    // Allow interrupts
again
    }
  }
}
void convert_coordinates(void)
  {
    //this routine converts the four ascii characters in the
    //x_char and y_char arrays into single integer words to
    //be sent to the servo PWM outputs

    x_coordinate = 0;
    x_coordinate = x_char[0]*1000 + x_char[1]*100 + x_char[2]*10 +
x_char[3]*1;

    y_coordinate = 0;
    y_coordinate = y_char[0]*1000 + y_char[1]*100 + y_char[2]*10 +
y_char[3]*1;

    }
void range_check(void)
  {
    //this routine checks whether or not the servo update values are
valid

    x_range_ok = false;
    y_range_ok = false;

    if ((x_coordinate >= 6290) && (x_coordinate <= 7075))
    {
      x_range_ok = true;
    }
    else
      x_range_ok = false;

    if ((y_coordinate >= 6290) && (y_coordinate <= 7075))
    {
      y_range_ok = true;
    }
    else
      y_range_ok = false;
  }
void update_servo_position(void)
```

```
  {
  //this routine updates the PWM value to change the servo positions
   if (x_range_ok == true)
   {
    CCR1 = x_coordinate;
    CCTL1 = OUTMOD_7;
    TACTL = TASSEL1 + MC_3;                    // Select ACLK source,
up-down mode, start timer
   }
    if (y_range_ok == true)
    {
      CCR2 = y_coordinate;
      CCTL2 = OUTMOD_7;
      TACTL = TASSEL1 + MC_3;                    // Select ACLK source,
up-down mode, start timer
    }

    TACTL = TASSEL1 + MC_3;                    // Select ACLK source,
up-down mode, start timer

    x_range_ok = false;
    y_range_ok = false;
  }

//-------------------------------------------------------------------
---------
// void InitSystem(void)
//
// This function configures the MSP430. Peripherals are initialized,
the
// external 8MHz crystal on LFXT1 is activated and used for ACLK and
MCLK,
// and the contents of the external EEPROM is validated.
//
// IN:  EEPROMImage[]
// OUT: -
//-------------------------------------------------------------------
---------
void InitSystem(void)
{
  WDTCTL = WDTPW + WDTHOLD;                    // Stop Watchdog timer

  // Port setup
  P1OUT = 0x00;                                // Clear output latch
  P1DIR = 0xfd;                                // All but P1.1 to
output
  P2OUT = 0x00;                                // Clear output latch
```

```
  P2DIR = 0xfb;                             // All but P2.2 to
output
  P3OUT = 0x00;                             // Clear output latch
  P3SEL = 0x3a;                             // Select I2C and UART0
pins
  P3DIR = 0xd4;                             // P3.0/1/3/5 inp,
others outp
  P4OUT = 0x00;                             // Clear output latch
  P4SEL = 0x55;                             // Timer funct. for
P4.0/2/4/6
  P4DIR = 0xaa;                             // P4.1/3/5/7 out,
others inp
  P5OUT = 0x00;                             // Clear output latch
  P5DIR = 0xff;                             // All output
  P6OUT = 0x00;                             // Clear output latch
  P6DIR = 0xff;                             // All output

                                            //void
initialize_servos(void)

                                            //  P1DIR |= 0x0C;//
P1.2 and P1.3 output
  P1SEL |= 0x0C;                          // P1.2 and P1.3 TA1/2
options
  CCR0 = 7860;                            // PWM Period set to 20ms
  CCTL1 = OUTMOD_7;                       // CCR1 toggle/set
                                          // CCR1 PWM duty cycle -
center servo
                                          //  7075 for 1ms
+pulsewidth
                                          //  6680 for 1.5ms
+pulsewidth - center
                                          //  6290 for 2ms
+pulsewidth
  CCR1 = 6680;                          //  center servo 1
                                          // CCR2 PWM duty cycle -
center servo
  CCTL2 = OUTMOD_7;                       // CCR2 toggle/set
  CCR2 = 6680;                           //  center servo 2
  TACTL = TASSEL1 + MC_3;                 // Select ACLK source, up-
down mode, start timer


  // Check if a button is held down during power-up. If so, trap
software here.
  // This is to allow the PC to invoke a BSL download, before the
MSP430 RESET
```

```
  // pin is switched to NMI mode, thus avoiding that any unintended
device
  // resets are caused by the TUSB3410 DTR signal.
  if (P4IN & 0x55)                          // Check all buttons
  {
    P4OUT = 0xaa;                           // All LEDs on
    while (1);                              // Loop forever
  }

  TUSB3410_RESET_LOW;                       // Hold TUSB3410 in
reset

  SVSCTL = 0xb0;                            // Enable SVS, monitor
only
  while ((SVSCTL & (SVSON + SVSOP)) != SVSON);  // Wait until SVS is
active,
                                            // and voltage
condition is met
  // Clock system setup
  BCSCTL1 |= XTS;                           // ACLK = LFXT1 = HF
XTAL
  __bic_SR_register(OSCOFF);                // Turn on XT1
oscillator

  do {
    IFG1 &= ~OFIFG;
    Delay100();                             // SW delay ~125us @
def. DCO
  } while (IFG1 & OFIFG);                   // Wait until XTAL is
stable

  BCSCTL2 |= SELM_3;                        // Select HF XTAL for
MCLK

  __enable_interrupt();                     // Global interrupt
enable

  InitI2C();                                // Configure USART0 for
I2C

  // Check if EEPROM is present. If so, verify EEPROM contents and
program
  // EEPROM image from MSP430 Flash memory if neccessary.
  if (!EEPROM_CheckNACK())
    if (!EEPROM_Verify(0, (void *)&EEPROMImage, sizeof EEPROMImage))
      EEPROM_Write(0, (void *)&EEPROMImage, sizeof EEPROMImage);
```

```
  WDTCTL = WDTPW + WDTHOLD + WDTNMI;            // Stop WDT, deactivate
RESET
  TUSB3410_RESET_HIGHZ;                         // Resume TUSB3410
operation

  InitUART();                                   // Configure USART0 for
UART
}
//-------------------------------------------------------------------
---------
// void InitUART(void)
//
// This function initializes the USART0 module for UART mode. It is
used to
// setup the communication link with the TUSB3410 controller. Using an
ACLK
// of 8MHz, the resulting UART baud rate is 460,800.
//
// IN:  -
// OUT: -
//-------------------------------------------------------------------
---------
void InitUART(void)
{
  U0CTL = 0;                                    // Ensure I2C mode is
disabled
  U0CTL = SWRST;                                // Hold USART logic in
reset
  ME1 |= UTXE0 + URXE0;                         // Enable USART0
TXD/RXD
  U0CTL |= CHAR;                                // 8-bit character
  U0TCTL = SSEL0;                               // UCLK = ACLK
  U0BR0 = 17;                                   // 8MHz / 460,800 =
17.xx
  U0BR1 = 0;
  U0MCTL = 0x52;                                // Modulation (See
User's Guide)
  U0CTL &= ~SWRST;                              // Release USART state
machine
}
//-------------------------------------------------------------------
---------
// void Delay100(void)
//
// This function implements a 100 cycle compiler optimization level
// independent software delay.
//
// IN:  -
```

```
// OUT: -
//-------------------------------------------------------------------
---------
void Delay100(void)
{
  asm("  mov.w    #33,R15  ");                  // Load loop counter
  asm("  dec.w    R15      ");                  // 1 Cycle
  asm("  jnz      $-2      ");                  // 2 Cycles
}
//-------------------------------------------------------------------
---------
// void InitI2C(void)
//
// This function initializes the USART0 module for I2C mode. It is used
to
// setup the communication link with the external EEPROM. Using an ACLK
of
// 8MHz, the resulting I2C bit rate is 100,000.
//
// IN:  -
// OUT: -
//-------------------------------------------------------------------
---------
void InitI2C(void)
{
  U0CTL = SWRST;                                // USART logic in reset
state
  U0CTL &= ~I2CEN;                              // Clear I2CEN bit
  U0CTL = I2C + SYNC + MST;                     // I2C master mode, 7-
bit addr
  I2CTCTL = I2CTRX + I2CSSEL_1;                 // Byte mode, repeat
mode, ACLK
  I2CSA = EEPROM_ADDRESS;                       // I2C slave address
  I2COA = OWN_ADDRESS;                          // Own address
  I2CPSC = 3;                                   // I2C prd = 4 * clock
prd
  I2CSCLH = 8;                                  // SCL high prd = 10 *
I2C prd
  I2CSCLL = 8;                                  // SCL low prd = 10 *
I2C prd
  U0CTL |= I2CEN;                               // Enable I2C module
operation
}
//-------------------------------------------------------------------
---------
unsigned char EEPROM_CheckNACK(void)
{
  while (I2CDCTL & I2CBUSY);                    // Wait for I2C module
```

```
  U0CTL &= ~I2CEN;
  I2CTCTL |= I2CRM;                         // Disable repeat mode
  U0CTL |= I2CEN;

  U0CTL |= MST;                             // I2C master mode
  I2CTCTL |= I2CTRX;                        // Transmit mode (R/W
bit = 0)
  I2CIFG &= ~NACKIFG;                       // Clear interrupt flag
  I2CTCTL |= I2CSTT;                        // Send slave address
  while (I2CTCTL & I2CSTT);                 // Start condition
  I2CTCTL |= I2CSTP;                        // Stop condition
  while (I2CDCTL & I2CBUSY);                // Wait until finished

  U0CTL &= ~I2CEN;
  I2CTCTL &= ~I2CRM;                        // Enable repeat mode
  U0CTL |= I2CEN;

  if (I2CIFG & NACKIFG)                     // NACK received?
    return true;                            // EEPROM not ready/not
detected
  else
    return false;                           // EEPROM detected
}
//-------------------------------------------------------------------
---------
// void EEPROM_ByteWrite(unsigned int Address, unsigned char Data)
//
// This function writes one byte to the specified address of an
externally
// connected EEPROM using the I2C module. After the write, it waits
until
// the EEPROM returns an ACK which indicates that the write was
completed.
//
// IN:  Address               // EEPROM address to write to
//      Data                  // 8-bit data
// OUT: -
//-------------------------------------------------------------------
---------
void EEPROM_ByteWrite(unsigned int Address, unsigned char Data)
{
  while (I2CDCTL & I2CBUSY);                // Wait for I2C module

  I2CBuffer[2] = Address >> 8;              // 16-bit address MSB
  I2CBuffer[1] = Address;                   // 16-bit address LSB
  I2CBuffer[0] = Data;
```

```
  I2CTxPtr = 2;                                 // Set I2CBuffer[]
pointer

  U0CTL |= MST;                                 // I2C master mode
  I2CTCTL |= I2CTRX;                            // Transmit mode (R/W
bit = 0)
  I2CIFG &= ~TXRDYIFG;
  I2CIE = TXRDYIE;                              // Transmit ready
interrupt
  I2CNDAT = 3;                                  // 1 control + 3 data
bytes
  I2CTCTL |= I2CSTT + I2CSTP;                   // Gen. start and stop
condition

  // Acknowledge Polling. The EEPROM will not acknowledge if a write
cycle is
  // in progress. It can be used to determine when a write cycle is
completed.
  while (EEPROM_CheckNACK());
}
//---------------------------------------------------------------------
---------
// unsigned char EEPROM_CurrentAddressRead(void)
//
// This function reads one byte of data from the current address of the
// EEPROM internal address pointer.
//
// IN:  -
// OUT: return()              8-bit data from EEPROM
//---------------------------------------------------------------------
---------
unsigned char EEPROM_CurrentAddressRead(void)
{
  while (I2CDCTL & I2CBUSY);                    // Wait for I2C module

  U0CTL |= MST;                                 // I2C master mode
  I2CTCTL &= ~I2CTRX;                           // Receive mode (R/W
bit = 1)
  I2CIFG &= ~RXRDYIFG;
  I2CIE = RXRDYIE;                              // Enable RX ready
interrupt
  I2CNDAT = 1;                                  // 1 data byte
  I2CIFG &= ~ARDYIFG;                           // Clear interrupt flag
  I2CTCTL |= I2CSTT + I2CSTP;                   // Gen. start and stop
condition
  while (!(I2CIFG & ARDYIFG));                  // Wait until RX is
finished
```

```
  return I2CBuffer[0];
}
//-------------------------------------------------------------------------
---------
// unsigned char EEPROM_RandomRead(unsigned int Address)
//
// This function reads one byte of data from the specified address of
the
// EEPROM internal address pointer.
//
// IN:  Address               EEPROM address to read from
// OUT: return()              8-bit data from EEPROM
//-------------------------------------------------------------------------
---------
unsigned char EEPROM_RandomRead(unsigned int Address)
{
  while (I2CDCTL & I2CBUSY);                    // Wait for I2C module

  I2CBuffer[1] = Address >> 8;                  // 16-bit address MSB
  I2CBuffer[0] = Address;                       // 16-bit address LSB
  I2CTxPtr = 1;                                 // Set I2CBuffer[]
pointer

  U0CTL |= MST;                                 // I2C master mode
  I2CTCTL |= I2CTRX;                            // Transmit mode (R/W
bit = 0)
  I2CIFG &= ~TXRDYIFG;
  I2CIE = TXRDYIE;                              // Transmit ready
interrupt
  I2CNDAT = 2;                                  // 1 control + 2 data
bytes
  I2CIFG &= ~ARDYIFG;                           // Clear interrupt flag
  I2CTCTL |= I2CSTT;                            // Gen. start condition
  while (!(I2CIFG & ARDYIFG));                  // Wait until address
was send

  I2CTCTL &= ~I2CTRX;                           // Receive mode (R/W
bit = 1)
  I2CIFG &= ~RXRDYIFG;
  I2CIE = RXRDYIE;                              // Enable RX ready
interrupt
  I2CNDAT = 1;                                  // 1 data byte
  I2CIFG &= ~ARDYIFG;                           // Clear interrupt flag
  I2CTCTL |= I2CSTT + I2CSTP;                   // Gen. start and stop
condition
  while (!(I2CIFG & ARDYIFG));                  // Wait until RX is
finished
```

```
  return I2CBuffer[0];
}
//-------------------------------------------------------------------
---------
// void EEPROM_Write(unsigned int Address, unsigned char *Buffer,
//                   unsigned int Count)
//
// This function writes a block of data to a specified address in the
// external EEPROM.
//
// IN:  Address               EEPROM start address
//      Buffer                Start address of data to write to EEPROM
//      Count                 Size of data block in bytes
// OUT: -
//-------------------------------------------------------------------
---------
void EEPROM_Write(unsigned int Address, unsigned char *Buffer,
                  unsigned int Count)
{
  unsigned int i;

  for(i = 0; i < Count; i++)
    EEPROM_ByteWrite(Address++, Buffer[i]);
}
//-------------------------------------------------------------------
---------
// void EEPROM_Read(unsigned int Address, unsigned char *Buffer,
//                  unsigned int Count)
//
// This function reads a block of data from the external EEPROM and
stores
// it to MSP430 memory.
//
// IN:  Address               EEPROM start address
//      Buffer                Start address of the buffer to store
EEPROM data
//      Count                 Size of data block in bytes
// OUT: -
//-------------------------------------------------------------------
---------
void EEPROM_Read(unsigned int Address, unsigned char *Buffer,
                         unsigned int Count)
{
  unsigned int i = 0;

  Buffer[i] = EEPROM_RandomRead(Address);

  for(i = 1; i < Count; i++)
```

```
    Buffer[i] = EEPROM_CurrentAddressRead();
}
//---------------------------------------------------------------------
---------
// unsigned int EEPROM_Verify(unsigned int Address, unsigned char
*Buffer,
//                            unsigned int Count)
//
// This function compares a block of data in the external EEPROM and
the MSP
// memory. The return value indicates whether the data is identical or
not.
//
// IN:  Address               EEPROM start address
//      Buffer                Start address of the buffer to store
EEPROM data
//      Count                 Size of data block in bytes
// OUT: return()              false on data mismatch, true on match
//---------------------------------------------------------------------
---------
unsigned int EEPROM_Verify(unsigned int Address, unsigned char *Buffer,
                           unsigned int Count)
{
  unsigned int i = 0;

  if (Buffer[i] != EEPROM_RandomRead(Address))
    return false;                               // EEPROM contents
mismatch

  for (i = 1; i < Count; i++)
    if (Buffer[i] != EEPROM_CurrentAddressRead())
      return false;                             // EEPROM contents
mismatch

  return true;                                  // EEPROM matches
buffer
}
//---------------------------------------------------------------------
---------
// void I2C_ISR(void)
//
// The I2C interrupt service function is used for data receiption and
// transmission. Global variables are used for data exchange.
//
// IN:  I2CBuffer[]
//      I2CTxPtr
// OUT: I2CBuffer[]
//      I2CTxPtr
```

```
//-------------------------------------------------------------------
---------
#pragma vector = USART0TX_VECTOR
__interrupt void I2C_ISR(void)
{
  switch (__even_in_range(I2CIV, 0x10))         // Use calculated
branching
  {
    case I2CIV_RXRDY :
      I2CBuffer[0] = I2CDRB;                    // Store RX data in
buffer
      break;
    case I2CIV_TXRDY :
      I2CDRB = I2CBuffer[I2CTxPtr];
      if (I2CTxPtr-- == 0)                      // TX finished?
        I2CIE &= ~TXRDYIE;                      // Disable TX ready
interrupt
      break;
  }
}
//-------------------------------------------------------------------
---------
// void TIMERB0_ISR(void)
//
// This function uses Timer_B7 to process SW1 push-button events.
// The timer is used in capture mode to generate an interrupt on the
initial
// button press. After that, compare mode is activated, and the status
of the
// button is polled every 50,000 ACLK cycles for debounce purposes.
//
// IN:  -
// OUT: ButtonState          Current status of SW1/SW2/SW3/SW4
//      ButtonSet            Flag register that indicates button press
//      ButtonReleased       Flag register that indicates button
release
//-------------------------------------------------------------------
---------
#pragma vector = TIMERB0_VECTOR
__interrupt void TIMERB0_ISR(void)
{
  if (TBCCTL0 & CAP)                            // In capture mode?
  {
    ButtonState |= 0x01;                        // Indicate SW1 button
press
    ButtonSet |= 0x01;
    ButtonReleased &= ~0x01;
    TBCCTL0 &= ~CAP;                            // Disable capture mode
```

```
    TBCCR0 += 50000;                          // Time to next compare
event
    __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
  }
  else if (!(TBCCTL0 & CCI))
  {
    ButtonState &= ~0x01;                     // Indicate SW1 button
release
    ButtonSet &= ~0x01;
    ButtonReleased |= 0x01;
    TBCCTL0 |= CAP;                           // Enable capture mode
    __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
  }
}
//-------------------------------------------------------------------
---------
// void TIMERB1_ISR(void)
//
// This function uses Timer_B7 to process SW2, SW3, and SW4 push-button
events.
// The timer is used in capture mode to generate an interrupt on the
initial
// button press. After that, compare mode is activated, and the status
of the
// button is polled every 50,000 ACLK cycles for debounce purposes.
//
// IN:  -
// OUT: ButtonState          Current status of SW1/SW2/SW3/SW4
//      ButtonSet            Flag register that indicates button press
//      ButtonReleased       Flag register that indicates button
release
//-------------------------------------------------------------------
---------
#pragma vector = TIMERB1_VECTOR
__interrupt void TIMERB1_ISR(void)
{
  switch (__even_in_range(TBIV, 0x0e))        // Use calculated
branching
  {
    case 0x04 :
      if (TBCCTL2 & CAP)                      // In capture mode?
      {
        ButtonState |= 0x02;                  // Indicate SW2 button
press
        ButtonSet |= 0x02;
        ButtonReleased &= ~0x02;
        TBCCTL2 &= ~CAP;                      // Disable capture mode
```

```
        TBCCR2 += 50000;                          // Time to next compare
event
        __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
      }
      else if (!(TBCCTL2 & CCI))
      {
        ButtonState &= ~0x02;                     // Indicate SW2 button
release
        ButtonSet &= ~0x02;
        ButtonReleased |= 0x02;
        TBCCTL2 |= CAP;                           // Enable capture mode
        __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
      }
      break;
    case 0x08 :
      if (TBCCTL4 & CAP)                          // In capture mode?
      {
        ButtonState |= 0x04;                      // Indicate SW3 button
press
        ButtonSet |= 0x04;
        ButtonReleased &= ~0x04;
        TBCCTL4 &= ~CAP;                          // Disable capture mode
        TBCCR4 += 50000;                          // Time to next compare
event
        __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
      }
      else if (!(TBCCTL4 & CCI))
      {
        ButtonState &= ~0x04;                     // Indicate SW3 button
release
        ButtonSet &= ~0x04;
        ButtonReleased |= 0x04;
        TBCCTL4 |= CAP;                           // Enable capture mode
        __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
      }
      break;
    case 0x0c :
      if (TBCCTL6 & CAP)                          // In capture mode?
      {
        ButtonState |= 0x08;                      // Indicate SW4 button
press
        ButtonSet |= 0x08;
        ButtonReleased &= ~0x08;
        TBCCTL6 &= ~CAP;                          // Disable capture mode
        TBCCR6 += 50000;                          // Time to next compare
event
        __bic_SR_register_on_exit(LPM0_bits);     // Wake-up CPU
      }
```

```
      else if (!(TBCCTL6 & CCI))
      {
        ButtonState &= ~0x08;                   // Indicate SW4 button
release
        ButtonSet &= ~0x08;
        ButtonReleased |= 0x08;
        TBCCTL6 |= CAP;                         // Enable capture mode
        __bic_SR_register_on_exit(LPM0_bits);   // Wake-up CPU
      }
      break;
  }
}
//-------------------------------------------------------------------------
---------
// void USART0RX_ISR(void)
//
// This function reads out a received character from the UART. After
that,
// the MSP430 returns active to process any pending events.
//
// IN:  -
// OUT: RxData                Received character
//      DataReceived          Flag to indicate that a char was RX'd
//-------------------------------------------------------------------------
---------
#pragma vector = UART0RX_VECTOR
__interrupt void USART0RX_ISR(void)
{
  RxData = U0RXBUF;
  DataReceived = true;
  __bic_SR_register_on_exit(LPM0_bits);         // Wake-up CPU
}
//-------------------------------------------------------------------------
---------
```

Appendix IX. Mechanical fabrication process

The mechanical portion of the system was designed using SOLIDWORKS™ three dimensional mechanical modeling software.

Fabrication of the gimbal components required the design and development of special machining fixtures. These fixtures were likewise developed and simulated using SOLIDWORKS™ and produced using traditional machining techniques. Fig. 1 depicts the first set of fixtures that are required:

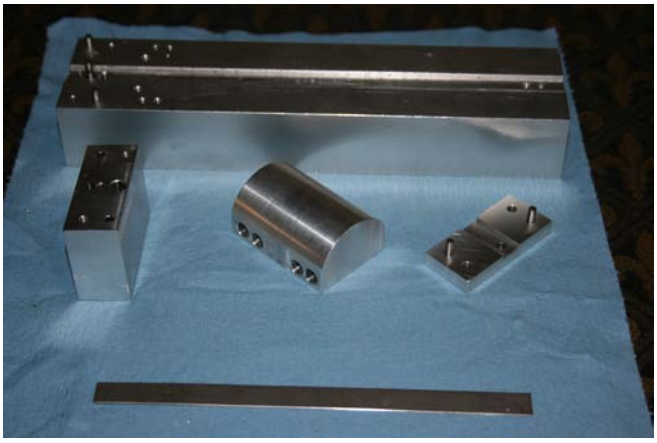


Fig. 1. Initial Gimbal fabrication fixture components

A blank piece of material is shown in the foreground of the Fig. 1. This material initially has a precision 0.125 inch guide hole and a #8 retention hole placed in one end

of the blank. The blank is then mounted into the slot of the fixture in the background. The guide hole in the blank fits over the precision dowel pin in the center of the slot on the left side and the retention screw holds the blank in place. Next, the gimbal slot is machined in the center of the blank using a 0.125 inch endmill cutting tool. The interior surfaces of the slot are crowned using a 0.0625 radius cutter. This is done to ensure that contact is made between the gimbal and the joystick actuator at a single point. This reduces operative friction and enhances smooth operation. After the slot is machined a series of holes are placed in the right hand side of the blank to enable pushrod connection in the final assembly. At this point the rectangular fixture block on the left is installed over the blank in the fixture slot. The half round fixture block is installed to the right of the rectangular block using two screws. The initial bend in the gimbal is then made manually, wrapping the blank up and over both the half round and rectangular fixture blocks. The material does exhibit significant springback and this necessitated the fabrication of a second, final forming fixture for the gimbal. This final forming fixture is shown in Fig. 2.

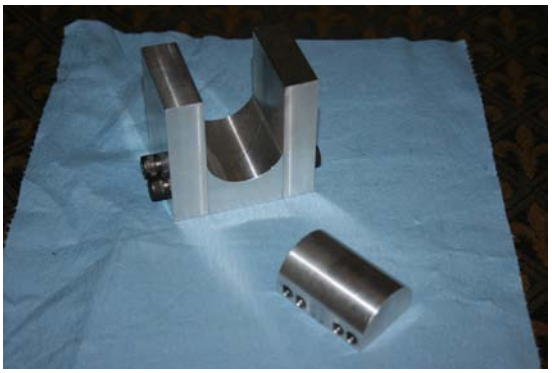


Fig. 2. Final gimbal forming fixture

The partially formed gimbal blank is then removed from the initial forming fixture and placed into the final forming fixture. The half round fixture block is reused and placed into the final forming fixture over the blank. A three ton arbor press is used to form the gimbal to final dimensions. Even with the significant material thickness employed in the body of the final forming fixture it was necessary to place a large C-clamp over the outside of the fixture while using the arbor press to prevent the fixture sides from bowing out. After final forming the partially completed part is replaced in the initial forming fixture and the small rectangular cap block is installed to hold the gimbal in alignment for machining the pivot holes. An undersize hole is then match drilled in both of the gimbal arms and reamed to a final size of 0.125 inches to accommodate the precision shoulder screws used to affix the gimbals to their mounting blocks.

The entire mechanical assembly was fabricated from 6061T6 aluminum with the exception of the gimbals and the mounting platform components. These components were made from a softer grade of aluminum in order to enable implementation of the necessary bends. Two gimbals were manufactured according to the preceeding process in addition to the mounting plates and servo mounts.

There were many fabrication challenges encountered in the course of the prototype development. The primary challenge throughout the design was the necessity of maintaining tight machining tolerances in order to minimize slop in the mechanism. The completed assembly of the initial mechanical prototype is depicted in Fig. 3.

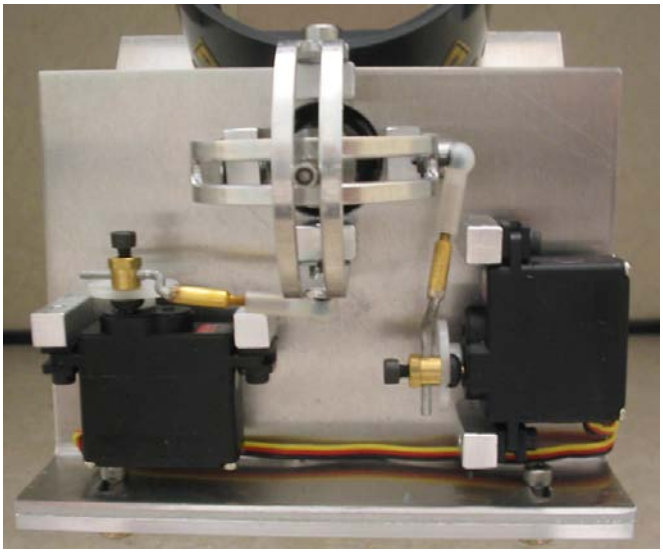


Fig. 3. Initial Prototype Medical robotic system Mechanical System