Design of Inertial Tracking System for Laparoscopic Instrument Trajectory Analysis

by

Calvin Gan

Electrical and Biomedical Engineering Design Project (4BI6)

Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario, Canada

Design of Inertial Tracking System for Laparoscopic Instrument Trajectory Analysis

by

Calvin Gan

Electrical and Biomedical Engineering
Faculty Advisor: Dr. Alexandru Patriciu

Electrical and Biomedical Engineering Project Report
submitted in partial fulfillment of the degree of
Bachelor of Engineering

McMaster University
Hamilton, Ontario, Canada
March 23, 2010

ABSTRACT

Laparoscopic surgery has brought about radical change in how surgery is performed today. The advantages of using tiny incisions to perform surgery are marred by very difficult and delicate techniques which must be applied by the surgeon. The need to track laparoscopic instruments results from the significant learning curve required to perform safe laparoscopic techniques, and the need to provide an objective assessment of the surgeon's skills. The idea is that through recording the instrument's trajectory, the surgeon can compare his or her movements with that of an expert. This provides an objective evaluation, in which the student can reassess and correct their movement as necessary. By attaching an inertial measuring unit (IMU) to the laparoscopic tool, relative positions and orientations can be calculated, and its path in 3-D can be approximated over time. The IMU data can be streamed to a PC where positions are written to an output file. Using graphing software (MATLAB used in this case) positions are plotted and the created trajectory is subsequently analysed for an objective assessment for comparison evaluations. The following report describes the process of acquiring the tool's trajectory by using inertial sensors, namely accelerometers and gyroscopes. Explanation of hardware and software design used to obtain position, orientation, and ultimately trajectory, along with experimental results are presented.

Keywords: laparoscopic surgery, minimally invasive surgery, inertial measuring unit (IMU), accelerometer, gyroscope, trajectory

ACKNOWLEDGEMENTS

The author would like to acknowledge and extend his gratitude to the following persons who have made the completion of this project possible:

Dr. Thomas Doyle, who served as project coordinator, for his continual advice throughout the entire 4BI6 course.

Jiten Mistry, the author's colleague and project member, for his effort and contributions put towards the project, and for his support and encouragement.

Dr. Alexandru Patriciu, who served as the author's faculty advisor, for his expertise in robotic applications. In addition to funding this project, Dr. Patriciu offered excellent guidance and feedback with many arising issues; without him this project would not be possible.

Once again, the author would like to thank all those who have helped him in any way over the course of completing the design project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCALTURE

Accelerometer: Sensor that outputs a signal (analog or digital) proportional to object's acceleration.

ADC: Analog to Digital Converter.

Angular Velocity: The rate of change of angular displacement with respect to time.

Drift: The ever-increasing difference between calculated locations from an inertial sensor to the actual location.

Gyroscope: Sensor that outputs signals (analog or digital) proportional to object's angular velocity.

IMU: Inertial Measuring Unit; device that measures an object's velocity (and position), and orientation, generally consisting of accelerometers and gyroscopes.

Inertial Sensors: Sensors that measure a change in motion such as acceleration or (angular) velocity.

Laparoscope: Key instrument used in laparoscopic surgery; camera is embedded at the tip where images are displayed on a monitor in the operating room.

PC: Personal Computer

Rotation: Angular motion about some axis.

Trajectory: Geometric path with time information.

Translation: Ability to move (change position) without rotating.

1 Introduction

1.1 Background

Laparoscopic surgery is a type of endoscopy, whereby the surgeon uses a specialized endoscope (called a laparoscope) and special laparoscopic tools to perform surgery through tiny (5-10mm diameter) incisions. This type of surgery is generally performed within the abdominal or pelvic cavity; inflexible instruments are most effective in this area and can reach organs of interest (e.g. gallbladder) fairly easily. Laparoscopic surgery (also known as minimally invasive surgery) provides a safe and successful surgical technique comparable to that of traditional surgical procedures. In some cases, for example in treatment of chronic cholecystitis, laparoscopic surgery is the preferred method [1], [3] due the advantages of using few small incisions. Many outcomes such as quick recovery, less scarring, and fewer post-operative complications result from laparoscopic techniques. In addition, this method can be applied for diagnoses as well as treatment, and as such, laparoscopic surgery is growing in frequency and importance. Laparoscopic surgery requires skills very different from those used in open surgery [2]. It requires precise hand-eye coordination, and because operations are done through a pivot point, motion is inverted (this is known as the "fulcrum effect"). Further, the operative field is displayed on 2D monitors via a laparoscope, which contribute to both reduced workspace and loss of depth perception. These complications are what make minimally invasive surgery such a difficult technique to master.

Owing to an increase in procedures, the degree of difficulty involved, and overall lack of experience, laparoscopic surgical accidents or malpractices have been increasing in number [3]. Shortage of experience or skills for laparoscopic surgery could stem from the fact that current methods of training do not include proper assessment of the surgeons' ability. Typically, resident surgeons begin their surgical education by observing their experienced counterparts in the operating room. They then contribute to the operation by performing basic techniques and/or diagnostic laparoscopy, where chances of causing damage to the patient are significantly reduced. Upon being certified as primary

surgeons, they can then perform a wide range of advanced laparoscopy. Throughout this process however, the skill of the surgeon is not precisely known. Evaluation (by an experienced surgeon for example) will contain subjective factors, and may be why current methods of training are potentially unsafe for the patient [4]. To improve on existing training procedures, it is necessary to develop some sort of objective, quantifiable assessment of manual skills in basic laparoscopic surgery.

For this design project, motion analysis is used to provide the objective assessment discussed. This requires tracking and recording motions of laparoscopic tools, where the data can be further analyzed. These recorded trajectories can be used in a variety of ways. For example, by comparing the motion acquired from an expert to that acquired from a novice (in a controlled trial), obvious differences should be seen; the novice can then make appropriate changes to his or her technique, resulting in a closer match between the two trajectories (and more skillful technique). The focus of this report will include the design of the tracking system itself.

1.2 Objectives

The objective of this project is to design an inertial tracking system aimed for laparoscopic instruments. While the surgeon moves the tool, the system will track and record 3-D positions in time, providing a means for objective assessment. The system should be lightweight and compact enough to be mounted on the surgical tool so it does not impede the normal operation of the surgeon. Such a system can be visualized in Figure 1, where the sensors are placed close to where the surgeon's hand would be. To provide some visual feedback and to get an idea of the performance of our tracking system, positions in all three-dimensions (one for each orthogonal axis) will be saved and plotted. As will be shown in experimental results, decent results are obtained for simple movements.

Accuracy when obtaining position and orientation through accelerometer and gyroscope sensors is, by nature, erroneous with time. This fact will be brought up later in the report,

and we shall see why inertial data is often complemented with other methods of tracking. The tracking system implemented (which is based only on inertial sensors) will be sufficient to show the general path of the user's motion, but cannot be 100% accurate. For this reason, a measure of accuracy (within 2mm for example) was not within our objectives. Rather, a measure of resolution was defined to be at 1ksamples/second. By increasing data points, a smoother trajectory (higher resolution) can be created which results in the original signal (motion) being reproduced as closely as possible. A drop in sampling rate on the other hand, may result in aliasing and the trajectory may look nothing like the surgeon's movement at all.

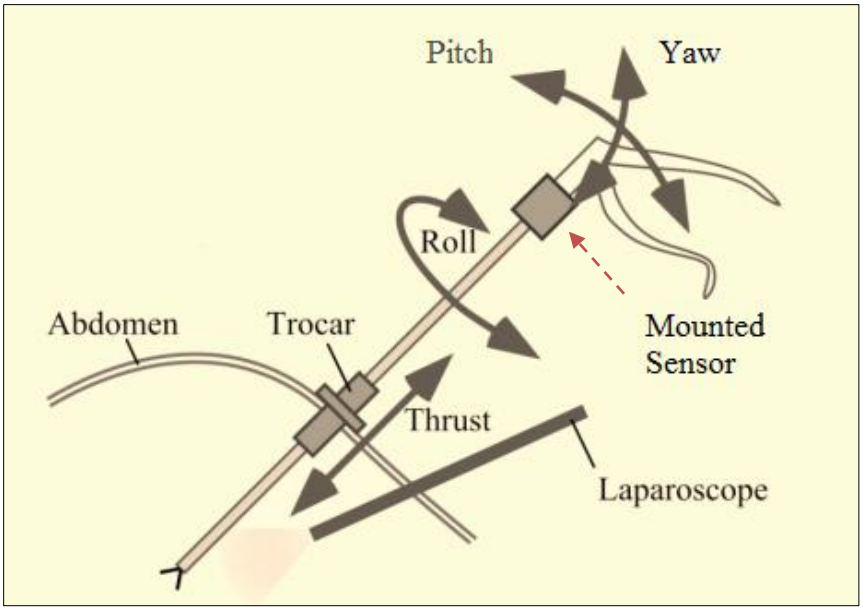


Figure 1: Laparoscopic instrument with mounted sensor[1]

1.3 General Approach to the Problem

The block diagram of the proposed tracking system is outlined in Figure 1.2. This approach consists of four components: the inertial measuring unit (IMU), microcontroller, USB controller, and PC.

[1] Image taken from [3]

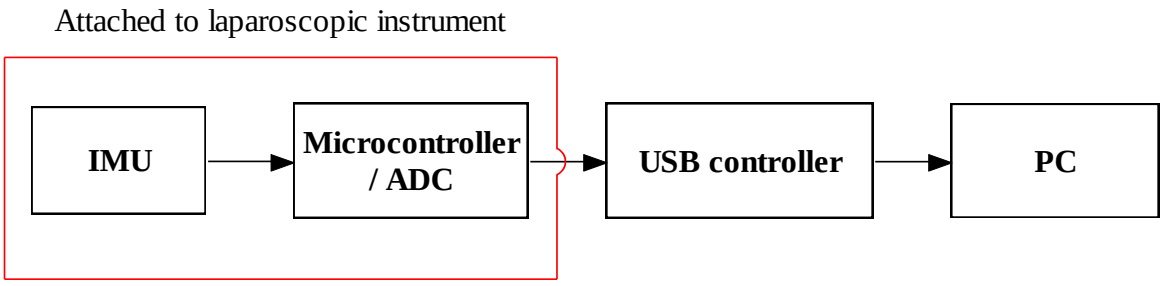


Figure 2: Block diagram of tracking system

The IMU consists of a set of analog sensors: a triple-axis accelerometer, a dual-axis gyroscope, and a single-axis gyroscope. Together, the three sensors allow us to measure six degrees of freedom (DOF). Analog outputs (six of them) from the inertial measuring unit (IMU) are fed into a microcontroller where the data is converted to a digital signal via the on board ADC. Depending on the signals produced by the sensors, or the voltage reference of the ADC, an amplification stage may be required to improve ADC performance and resolution. In developing stages, an amplification stage was used to slightly increase ADC performance; however this amplification system won't be used in the final product. See Appendix A for design details. Once digitized, the data is processed to calculate position and orientation, which is also done at the microcontroller stage.

Very briefly, the process of estimating positions and orientations are as follows: Through integration (over time) of angular velocity provided by the gyroscopes, we are able to obtain the relative angles $\theta_x, \theta_y, \theta_z$ about each axis X, Y and Z respectively. Through double integration of acceleration provided by the accelerometer, we obtain the relative distance moved by each axis (X, Y and Z), and hence acquire position. It is important to realize the IMU, being mounted on the laparoscopic tool, will provide data in a *moving* frame, but what we really need are positions in a *base*, or *global* frame of reference. Through linear algebra (robotics), transformation via rotation matrices will resolve this issue. This can be visualized in Figure 1.3. More detail is provided in later sections.

Lastly (going back to Figure 1.2), because the IMU is intended to be mounted on the laparoscopic instrument, data must be streamed over to the PC in order to be recorded,

saved, and analyzed. This is done through a USB controller, detached from the laparoscopic tool.

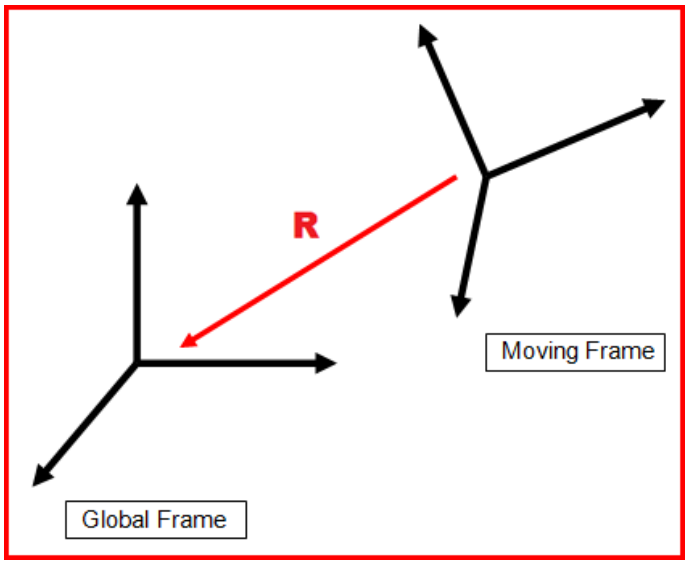


Figure 3: Rotation applied to moving frame in order to achieve coordinates with respect to a base frame

1.4 Scope of the Project

The scope of this project is to implement a motion tracking system using an IMU consisting of accelerometer and gyroscope sensors. The resulting data is then sent to a PC via USB controller for analysis. This report encompasses the design of IMU and processing the raw data (i.e. the first two blocks shown in Figure 1.2); it will not discuss implementing the USB controller or the comparative analysis done at the computer stages. As stated previously, the application of a tracking system can be used to provide objective assessment during laparoscopic training, as well as reduce the steep learning curve associated with this type of surgery.

The extent of this project is mainly limited by the accuracy of trajectory produced. As brought up earlier, strictly using inertial sensors cannot guarantee a quantity of accuracy or error at any point in time. In order to approximate position (X, Y, and Z coordinates) and orientation ($\theta_x, \theta_y, \theta_z$ - or roll, pitch, yaw), discrete integration needs to be applied. For each integration interval, error terms (or constants) are introduced and accumulated,

thereby decreasing accuracy as the integration interval (time) increases. This effect is even more prevalent when calculating positions from acceleration. Within seconds the data may be full of errors, rendering it unusable. It is for this reason that this project is intended to obtain the general trajectory within a certain period of time (seconds rather than minutes/hours).

Increasing accuracy chiefly depends on the integration method, sampling rate, and ADC resolution. Also, to account for the inherent errors present in inertial sensors, methods to drastically reduce inaccuracies will be discussed (such as combining with additional tracking systems).

2 Literature Review

2.1 Current Systems for Tracking Laparoscopic Instruments

Many laparoscopic tracking systems have been designed for reasons explained earlier; that is, by analyzing motions of the instrument, laparoscopic skills can be assessed. There are several devices commercially available or are under development that aims to track movements of laparoscopic instruments. These systems employ a variety of tracking methods which include optical, mechanical, acoustic, or electromagnetic technology. Before discussing about current tracking systems, said methods will be summarized and compared.

2.1.1 Methods

Optical tracking uses cameras or position-sensitive optical devices to track IRED (infrared emitting diode) signals. This type of position tracker works by placing cameras at fixed points which detect a set of IREDs mounted on the object. Mechanical tracking systems rely on a direct mechanical connection between a reference point and the object. The connection is usually the form of an arm, and the system detects movements through the arm. Acoustic devices emit and sense ultrasonic sound waves to determine the position and orientation of the object. The sensors may be stationary, while the ultrasonic emitters are mounted on the object. The system measures the length of time it takes for the sound to travel from the object to the sensors, thereby calculating its distance. Lastly, electromagnetic tracking systems function by measuring the magnetic field strength generated by three mutually orthogonal coiled wires. When current is applied through the wires (which are attached to object), magnetic fields are generated and can be measured by a sensor at some fixed location. Position and orientation of the object can therefore be determined.

Table 2.1 below compares the advantages and disadvantages of each method described.

Table 1: Comparing common tracking system methods

Technology	Advantages	Disadvantages
Optical	Reasonable range, accuracy, and resolution	Suffers from line-of-sight problems (i.e. camera needs to be able to "see" IREDs at all times)
Mechanical	Very good accuracy, resolution; interference immunity	Extremely limited range; constrained motion due to physical connection
Acoustic	Large range at low cost	Slow update rates (speed of sound relatively slow); speed of sound affected by environment (i.e. temperature, humidity, etc.)
Electromagnetic	Low latency; no line of sight problems	Affected by distortions in magnetic field caused by metal objects; rapid decrease in accuracy/resolution with distance

2.1.2 Tracking Systems

A number of important aspects are considered when designing laparoscopic tracking systems. Some issues are portability, available haptic feedback, and accuracy. In addition, there are as many as three environments in which tracking may occur: in a virtual reality trainer, a box trainer, or in the operating room. The following paragraphs will briefly discuss some tracking systems, while covering each environment mentioned. The technology employed in each system and advantages and disadvantages will also be highlighted.

The **Xitact IHP – Instrument Haptic Port** is a portable virtual reality (VR) system designed to track the motion of a surgical instrument. Produced by Xitact S.A (Xitact S.A. Morges, Switzerland, http://www.xitact.com), this tracking system provides force feedback and allows the use of real surgical instruments with handles and graspers (see Figure 4 below). To measure position and orientation, optical encoders are placed on both the Lin/Rot (linear and rotational drive), which is attached to the instrument, and the

pantoscope, which is attached to the base of the system. Connection to PC via USB permits simulation through software. Although VR systems provide a means of accurately tracking instruments, current issues include the realistic (or unrealistic) behavior of haptic feedback offered. Studies suggest that force feedback provided in VR trainers is far from that by real laparoscopic instruments experienced in an operating room or a box trainer [4].

Figure 4: Xitact IHP Tracking and Haptic System[2]

An ultrasound wireless positioning system developed at Delft University of Technology in The Netherlands, gives the exact 3-D location and orientation of the instruments in the patient by using acoustic tracking. Mounting ultrasound transmitters onto the instrument and placing ultrasound receivers above the patient (as seen in Figure 5) create an interference-free environment. That is, the surgeon is not hindered by any wires or mechanical devices, and the line-of-sight from marker to transmitters is not obstructed. The design of this system allows it to be used in the operating room, which encompasses patient safety (i.e. no nearby electrical wires) and ergonomics (how surgery will be actually performed). Additional advantages of operating room tracking systems include the use of real laparoscopic tools and the sense of true force feedback (not a virtual feedback). Like all acoustic systems, the ultrasound wireless positioning system suffers

[2] http://www.xitact.com

from environmental effects; however, if operating room conditions are kept fairly constant, this weakness will have little effect.

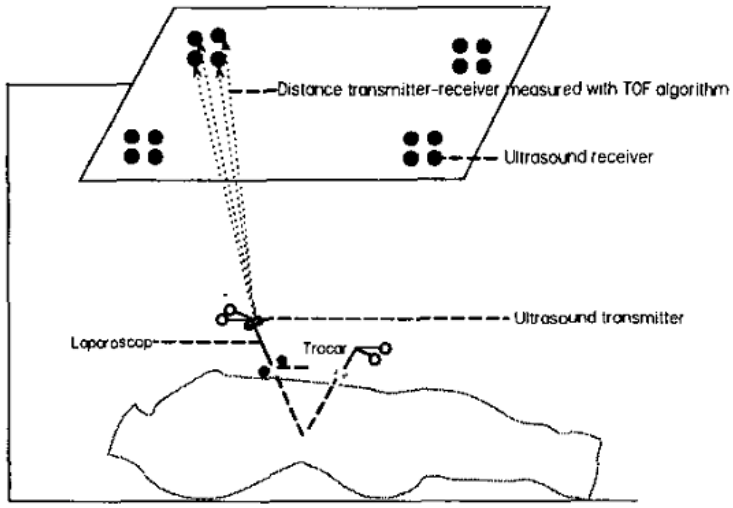


Figure 5: Ultrasonic tracking system: receivers situated above the surgical table and transmitters on surgical instruments [7]

The final tracking system discussed will be the **BlueDRAGON** tracking system. Two four-bar passive mechanisms attached to the laparoscopic tool, and position sensors integrated into the mechanisms' joint measure position and orientation (see Figure 6). Additionally, force/torque and contact sensors are implemented into the system such that grasping forces and tool/tissue contact forces can be measured. Real laparoscopic tools are used, thus providing natural haptic feedback. This mechanical tracking system is bulky, and while it can be used in the operating room, the BlueDRAGON can also be used in a box trainer environment. In a box trainer environment, a realistic environment is also presented where resident surgeons can further develop their techniques before moving on to operate on real patients.

For now, it seems the majority of trackers occupy the realm of virtual reality. Through research we found an additional 13 systems (to the three mentioned above), most of which can be used in the virtual environment. Despite lack of "real" haptic feedback, VR systems provide very accurate measurements, and can implement ways of tracking grasping techniques (i.e. closing and opening of instrument handles). Methods such as

acoustic or inertial systems have no way of tracking this additional degree of freedom. So far, not many laparoscopic tracking systems make use of inertial sensors. This is probably due to accumulated errors and other inherent factors. However, in the next section we'll see how inertial measuring units can play key roles in navigational systems and why we believe it is worth tracking with inertial sensors.

Figure 6: CAD drawing of the BlueDRAGON system [8]

2.2 Methods and Limitations of Inertial Measurement Systems

Inertial systems are a different type of mechanical tracking system (electromechanical), which rely on the principle of conservation of angular momentum. Gyroscopes are used to directly measure angular velocity, which can be integrated to achieve orientation (angles). Coupled with accelerometers (a position tracking device), the system has the ability to tack up to six degrees of freedom. The use of inertial sensors implemented in our design is to accomplish exactly this. Motivation for using IMUs include: fast data rates, low power consumption, light in weight, small packaging, no line-of-sight problems, and their relatively cheap cost. As stated before, inertial sensors are prone to errors resulting from drift and noise, and within seconds data can become useless. Despite these shortcomings, inertial sensors play a key part in navigational systems on many vehicles like aircrafts and submarines. In some cases, IMUs are implemented within HMD (head-mounted displays) for use in simulators or helmet tracking. Inertial

sensors are able to track for long periods of time with great accuracies in these cases because they are complemented with another form of tracking device.

In regards to aircraft navigation, inertial sensors are often tied with GPS or satellite positioning systems. Short term data is supplied by the inertial system, while accumulated errors are corrected by the satellite positioning system. An example of an HMD which implements inertial sensors is the Ascension Hy-BIRD (Figure 7). This design fuses optical and inertial technologies, providing (seamlessly) continuous tracking. The IMU in this case provides data in between optical frames and/or when the optical scanner is obstructed. In turn, the optical system must supply updated positions/orientation to the IMU because of the constant drift that occurs.

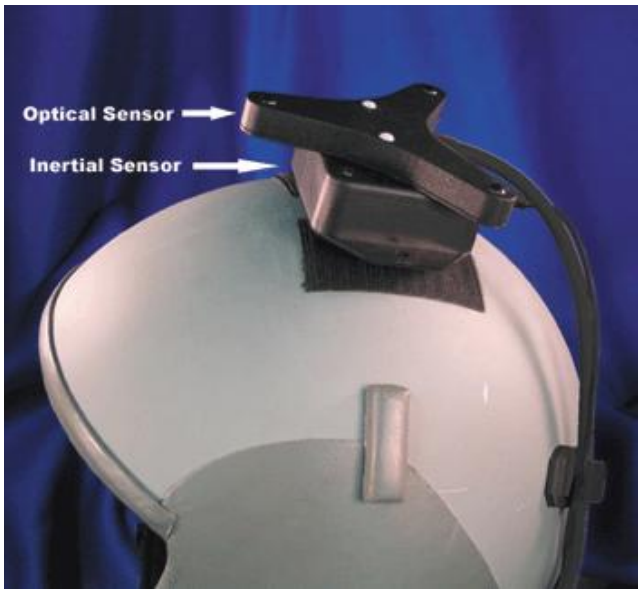


Figure 7: Ascension Hy-BIRD from Inition[3]

As discussed, an inertial system in the absence of another tracking system is subject to many factors that jeopardize accuracy. It is usually required that some form of feedback and update system is also incorporated into the overall tracking system. The purpose of this project is to track purely on inertial sensors, in hopes to later incorporate the system with another tracking method.

[3] http://www.inition.co.uk/

3 Statement of Problem and Methodology of Solution

3.1 Statement of Problem

In this project, we implement the IMU's ability to (indirectly) acquire position and orientation for use in a tracking system. The IMU, consisting of an accelerometer and gyroscopes, has low power consumption, and is very compact. This is ideal for mounting it, say to a laparoscopic tool (*cf.* Figure 1 on page 3) for tracking purposes. What remains is to compute desirable values in order to recreate the user's motions. This data can subsequently be used for comparative analysis or for assessment purposes.

3.2 Linear Accelerometers and Gyroscopes

An accelerometer is a device that measures acceleration forces. These forces could be static (gravity) or dynamic (movement and vibrations). Technically, there are two types of accelerometers: linear and angular accelerometers[4]; generally, when we talk about accelerometers, we refer to those that measure linear acceleration. Since the scope of this project only surrounds linear accelerometers, this will be our focus.

Different types of accelerometers exist. One method to create accelerometers is through the piezoelectric effect, whereby microscopic crystal structures produce a voltage upon being stressed due to acceleration forces. An alternative way includes measuring changes in electrical capacitance (see Figure 8 page 14). Acceleration deflects the moving mass and unbalances the differential capacitor resulting in a sensor output whose amplitude is proportional to acceleration.

In addition to the way they are created, accelerometers can differ by their output signal. Digital and analog accelerometers exist, where the former provides a discrete binary code proportional to its input (acceleration), and the latter provides a voltage that is proportional to its input (acceleration).

[4] Angular accelerometers measure the rate of change of angular rotation/velocity. These types of accelerometers are rarely used.

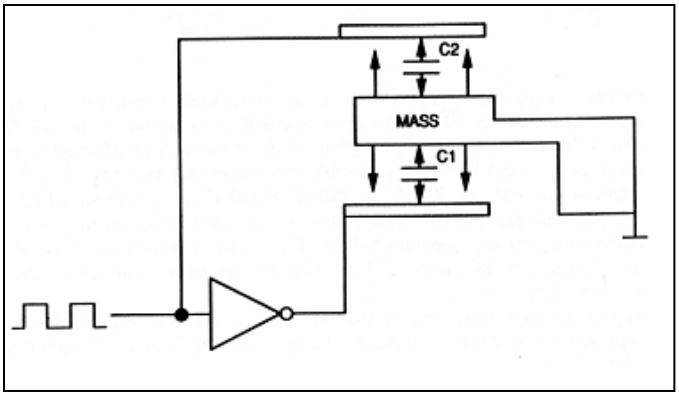


Figure 8: Differential capacitive accelerometer

Gyroscope sensors are also called angular rate sensors, as they measure the rate of rotation (rotational speed). As the case with accelerometers, various types of gyroscopes exist. Theoretical details will not be of concern here, but basically there are two main types: The MEMS (micro electro-mechanical) rate sensors are designed to measure angular rate via the Coriolis force, and FOG (fibre optic gyro) rate sensors operate using a fibre optic ring and a solid-state laser to measure rotation rates using the Sagnac effect[5]. The gyroscopes chosen for our system is of the "Coriolis principle" type.

Gyroscope sensors can also be analog or digital; and, depending on the type of microcontroller you are interfacing with, you may or may not have a choice.

In addition, there are options for single, dual, or triple-axis sensors. You can always form dual or triple-axis sensors by combining two or three single-axis sensors by positioning them so their axis directions are mutually orthogonal.

3.3 Methodology of Solution

The selection of microcontroller and inertial measuring unit (IMU) will be discussed in the next chapter (i.e. Experimental & Design Procedures). For now, assume we have an IMU consisting of a three-axis accelerometer and three-axis gyroscope, and a microcontroller to do the processing. Output data from the IMU are collected as

[5] http://www.xbow.com/support/Support_pdf_files/RateSensorAppNote.pdf

proportional acceleration values and proportional angular velocity values as described previously. Figure 9 and Figure 10 below shows the gyroscope and accelerometer sensor values after they've been digitized. As seen, every axis starts off with some sort of offset, known as the "bias". This bias value can cause incorrect readings if uncompensated for, as we'll see shortly.

In Figure 9, rotations about the Z, Y, and X axis can be seen by their variations along the bias value. What's seen here, is that by only rotating along the IMU's (the gyroscope's, specifically) Z-axis, the X and Y output are unaffected. Similarly, rotation about X or Y does not affect other axes. This shows that each axis is placed orthogonal to each other, as desired. Also, because only rotation was applied, variations in accelerometer values should be fairly constant, as reflected in Figure 10.

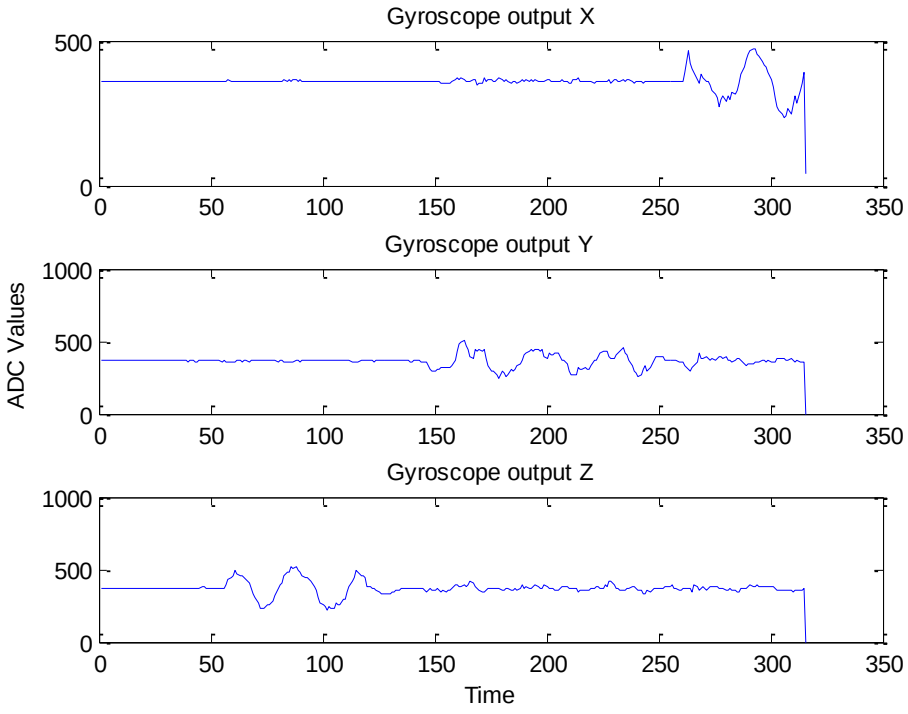


Figure 9: Raw gyroscope output

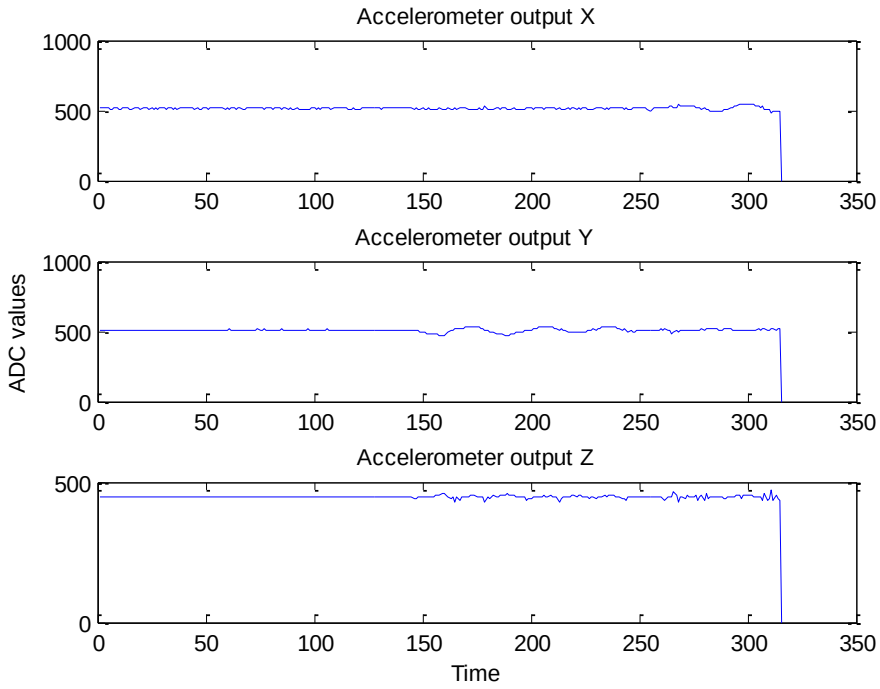


Figure 10: Raw accelerometer outputs

Now that we have proportional accelerations and angular velocities, integrating these values will achieve position and angles. However, in order to obtain negative values, it is clear we must subtract the correct bias values from each axis. The result of doing so with the gyroscope outputs is seen in Figure 11.

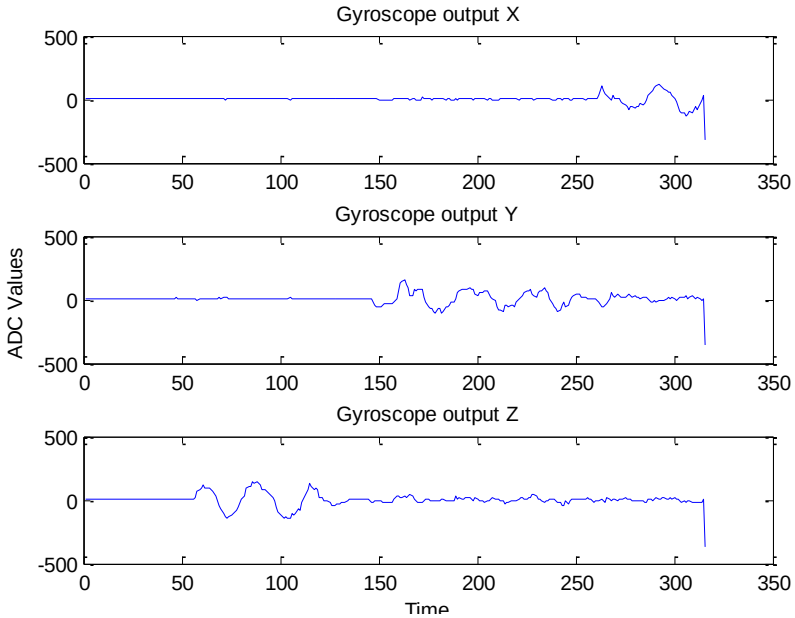


Figure 11: Raw gyroscope output post subtraction

Only after subtraction of this bias value, can we get negative and positive readings, which we then perform integration on. It is clear that if incorrect subtraction occurs, the “area under the curve” will be affected, producing undesirable values. Also, because integration is discrete, choice of integration method will also affect overall results. The fact that we employ discrete integration results in the accumulated errors discussed before.

One last issue will be brought up in this section. That is the notion of the IMU being in a moving frame. Because the IMU itself is moving, position updates are given with respect to this frame, but we need them defined in a universal or global frame of reference. Calculating a new rotation matrix after every interval creates the transformation which solves this issue. The rotation matrix can be thought of as an operator, where if multiplied by a vector, will transform the vector to a desired reference frame.

4 Experimental & Design Procedures

4.1 Hardware

To fully describe the 3-D motion of laparoscopic instrument movement, we require six measurements: $\theta_x, \theta_y, \theta_z$ and X, Y, Z. From this specification, it is required to obtain an IMU able to measure six degrees of freedom (DOF). Before we attempt this step, it is first important to select the microcontroller with which we are interfacing with. As mentioned last section, should our choice of microcontroller not have on board ADC modules to accept analog inputs, then digital sensors are required. The microcontroller we went with was the ATmega328, mainly because the Arduino Duemilanove is based on this chip. The Arduino Duemilanove is a microcontroller board which allows programming of the Atmega328 microcontroller through a USB interface. Available from this board are six analog input pins, 14 digital I/O pins, supply voltages 3.3V and 5V, and of course GND (0V). A diagram of the Arduino Duemilanove is shown below (Figure 12).

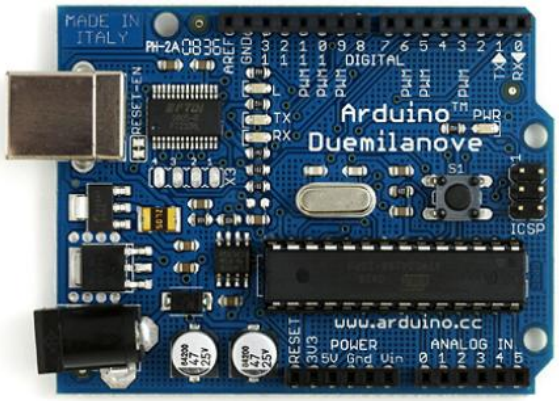


Figure 12: Arduino Duemilanove ("2009")[6]

When selecting the IMU, it was imperative to get a three-axis accelerometer and three-axis gyroscope as discussed earlier. An IMU produced by Sparkfun provided a very compact three-axis analog accelerometer along with two analog gyroscopes. Together,

[6] http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

the single-axis gyroscope and dual-axis gyroscope allowed for measurements about all three X, Y, Z axes. The unit can be seen in Figure 13, where the gyroscopes are the two larger packages neighboring (dual-axis gyro providing pitch and roll on the left; single-axis providing yaw on the right) the central three-axis accelerometer in the bottom view.

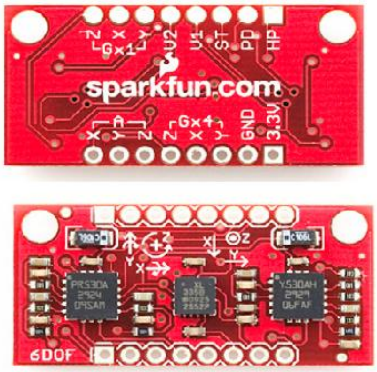


Figure 13: 6 DOF IMU from Sparkfun[7]

Connections from IMU to the Arduino board are shown in Figure 14. Six analog pins are used, one for each degree of freedom measured by the inertial unit. Voltage supplied to the IMU is 3.3V. No amplification or filtering was necessary, as typical signal outputs from the IMU are shown in Table 3 (with V_{in}=3.3V).

Table 2: IMU sensor measurement range

Sensors in IMU:	Measurement Range (1X)	Measurement Range (4X)
LPR530AL (pitch and roll gyro)	±1200°/s	±300°/s
ADXL335 (triple-axis accelerometer)	±3g	
LY530ALH (yaw gyro)	±1200°/s	±300°/s

Notes: - Gyroscope has an amplified (4X) and non-amplified (1x) output

- Accelerometer only has one output with given measurement range

Table 2 above describes the measurement range for each sensor. These ranges are used to calculate the maximum and minimum voltage ranges calculated in Table 3 (page 20).

[7] http://www.sparkfun.com/commerce/product_info.php?products_id=9431

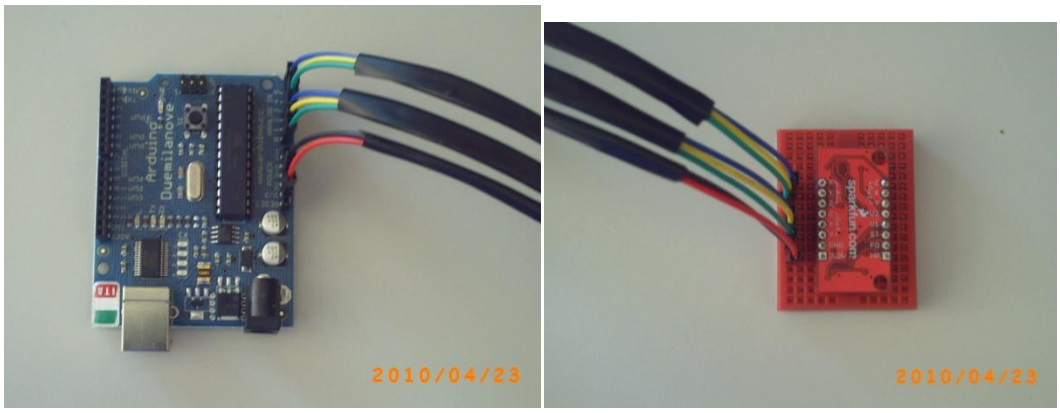


Figure 14: Arduino Board and IMU connections

Table 3: Accelerometer and Gyroscope output voltages

	$AccX_{out}$	$AccY_{out}$	$AccZ_{out}$	$GyroX_{out}$	$GyroY_{out}$	$GyroZ_{out}$
V	1.65V ±330mV/g	1.65V ±330mV/g	1.80V ±330mV/g	1.23V ±3.33mV/°/s	1.23V ±3.33mV/°/s	1.23V ±3.33mV/°/s
V_{max}	2.65 V	2.65 V	3.78 V	2.229 V	2.229 V	229 V
V_{min}	0.66 V	0.66 V	0.81 V	0.231 V	0.231 V	229 V

Note: While using IMU, the 4X amplified output was used

The voltages were all reasonable (i.e. not in the mV range) values, and we found no reason to amplify signals. To see a potential amplification system to be used if necessary, see Appendix A.

Table 4: IMU power consumption

	$V_{dd,}$	I_{dd}	P
LPR530AL (pitch and roll gyro)	3.3 V	6.8 mA	22.44 mW
ADXL335 (triple-axis accelerometer)	3.3 V	375 µA	1.24 mW
LY530ALH (yaw gyro)	3.3 V	5.5 mA	18.15 mW

Notes: I_{dd} was calculated as a max (i.e. when V_{dd} = 3.6V)

Max power = 41.83 mW consumed

From the Arduino Duemilanove to PC, the Serial Peripheral Interface (SPI) connection shall be used.

3.2 Software

Now that the general methodology in finding position and orientation from accelerometer and gyroscope data has been discussed, this section will provide flow charts and software design that helped us complete this project.

Programming of the microcontroller (ATmega328) is done through Arduino software (an open-source environment). Programs that use this software are written in the Arduino language, which is based on C/C++; as such, it was fairly easy to get familiarized with the language. A reason for choosing the Arduino Duemilanove was its user-friendly interface, allowing programs to be easily verified and quickly uploaded to the ATmega328 microcontroller.

When working with analog signals, the topic of sampling rate and ADC resolution tends to surface. The ATmega328 has a 10-bit ADC, and the reference voltage can be adjusted through software by sending a desired signal in pin AREF. By varying this voltage as necessary, ADC performance increases, and the quantization error present in all analog to digital converters is be decreased (but not eliminated). For our project, we provided the AREF pin with 3.3 volts. This means each count (ADC value) had a voltage of: $ADC_{count} = \frac{3.3\,V}{2^{10}-1} \cong 3.23mV$.

From online references, the sampling rate of reading in one analog input is ~100μs, good enough for 10 KHz sample frequency. Though there are ways (e.g. including define parameters) to further increase the rate, 10 KHz is sufficient for this project. The determining factor occurs when data is written through the USB (serial) port to PC. In order to stream data for analysis, at least three position variables must be written. After the program was written (which can be seen in Appendix B), each iteration (reading and writing six variables) ends up taking 0.04s. This is nowhere near 1ksamples/s like intended; nonetheless it should still give passable results – that is, the *general* trajectory of a user's movement can be recreated. These results are shown in the next section.

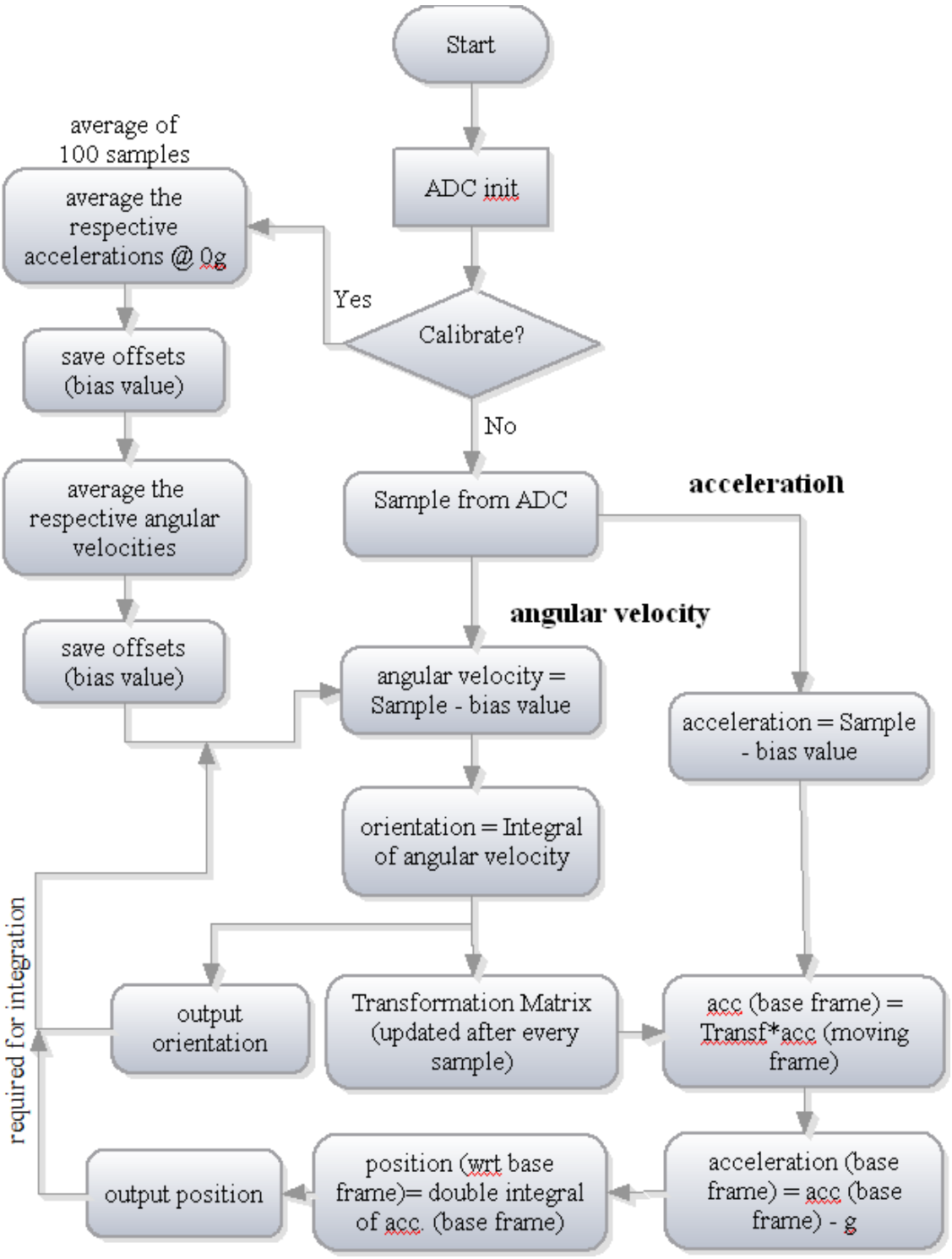


Figure 15: Implemented algorithm solving for position and orientation given data from accelerometer and gyroscope

Figure 15 above is a flow chart of the algorithm implemented in developing stages. Some explanation to selected stages will be clarified.

To begin with, a calibration function was created which involves averaging samples to calculate the bias value. The bias value is important (mentioned in section 3.3) in order to achieve correct negative and positive values. The importance of an accurate bias value is why averaging (here, 100) a sample was implemented. The calibration is only done once, at the very start of the program.

The method of integration used was the trapezoidal method, implemented as follows:

$$current\ angle = prev.angle + (\frac{prev.angular\ vel. + current.angular\ vel.}{2})dt$$

This is applied once to angular velocity, and twice to acceleration.

After calculating angles from the gyroscope output (via integration just mentioned), a rotation matrix is formed by the following formula:

$$R = R_x(angleX)R_y(angleY)R_z(angleZ)$$

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{bmatrix}, R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\alpha & 0 & cos\beta \end{bmatrix},$$

$$R_z(\alpha) = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here, R (the multiplication of all matrices) is the general rotation matrix which will be used to transform the acceleration.

To find acceleration in base frame, that is, to transform the acceleration:

$$a_{base}^{k} = R_{base}^{t_{k+1}} a_m^k$$

Where $R_{base}^{t_{k+1}} = R_o R_{t_0}^{t_1} \cdots R_{t_k}^{t_k+1}$ (post multiplication)

The super/subscript “t” represents time, equivalent to one pass of the entire flow chart in Figure 15 above. To clarify, the “start” is “Sample from ADC”, and the “end” is when positions and orientations are calculated and outputted.

5 Results and Discussion

To show results, a program was created to write orientations ($\theta_y, \theta_x, \theta_z$) and positions (X, Y, and Z) from the microcontroller to the PC via USB (serial connection). Simple motions were formed, all within 10-15 seconds to reduce effects of accumulated errors. Even then, results were not accurate. The following diagrams were created from MATLAB via a text file whose first three columns are gyroscope outputs $\theta_y, \theta_x, \theta_z$ and last three columns positional outputs X, Y, Z. An example of the output file is shown in Appendix B. Positions are plotted only, which have units of cm. Orientations are not plotted, but have units of degrees.

The IMU was not mounted on any sort of instrument; rather, as shown in Figure 14 (page 20), motions were made by simply moving the breadboard which the IMU is attached to. One issue that impeded free movement were the wires connecting the IMU to the Arduino board. These wires sort of had a negative effect in whatever direction motion was because they were quite stiff. It's hard to say whether this had an overwhelming effect with the resulting trajectories, but movement did feel restricted.

In my implementation of positions, I did not compensate for gravity (besides the initial orientation). What this means, is that if the IMU is placed with an initial orientation, and after the program starts, is left at that initial orientation, all positions/orientations will read zero as expected. However, performing **only** rotation such that gravity ($9.8 m/s^2$) now affects some other vector, the accelerometer will sense acceleration along that vector, and the program will calculate that it's moving, when it is actually not. So while testing, it was important that the IMU be kept at one orientation the entire time (usually flat on the surface as in Figure 14 (page 20)).

Following are a few tracking motions created by the method described above. Again, simple movements were the key to ensure the system could even track to some degree.

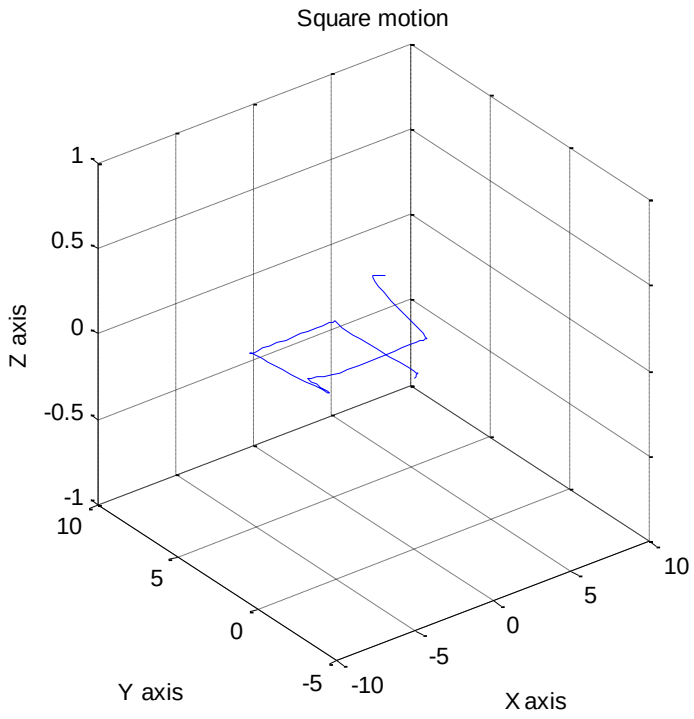


Figure 16: Square motion

Figure 16 above shows the recreated path after the user (me) made a square/rectangular movement in the X-Y plane with the inertial measuring unit. Note that there should be no Z displacement as I did not lift the IMU (and although it may look like there is, there isn't any).

Viewing the same data in the X-Y plane is seen in Figure 17 below. The tiny spike at around X=-4, Y=1 (below graph) was evidently not intended, yet still shows in the graph. The path begins at point (0, 0) and ends around (5, 6.9).

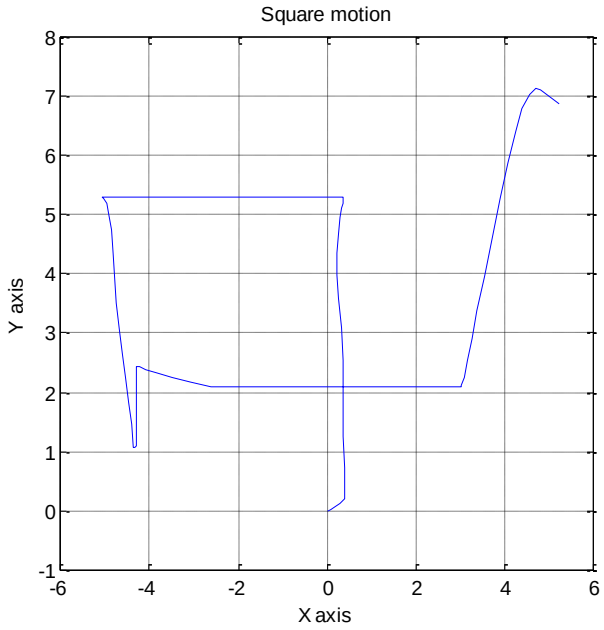


Figure 17: Square motion in X-Y plane

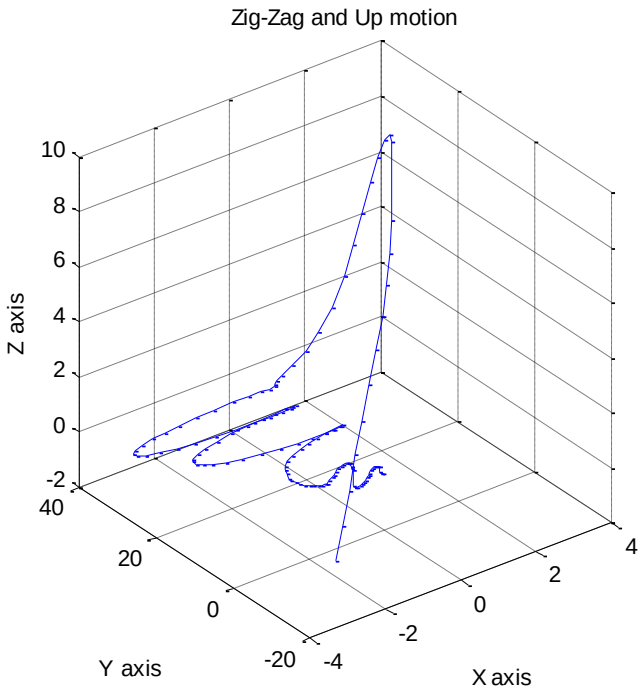


Figure 18: Path of zigzag motion, followed by an up and downwards motion

The motion here starts at the zigzag area (0, 0, 0). The general movement was a left-right repeating motion, then a translation in the positive Z direction. Finally, the IMU was put back to the starting point. The tracking system seems to capture the motion with decency. The graph actually plots the Y-direction range to be about 30cm. This was not the case in the real motion formed, which was about 10cm max.

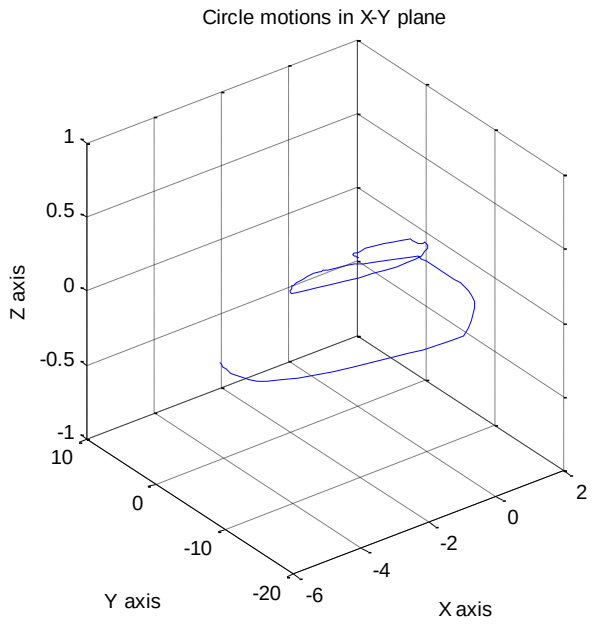


Figure 19: Circular movements

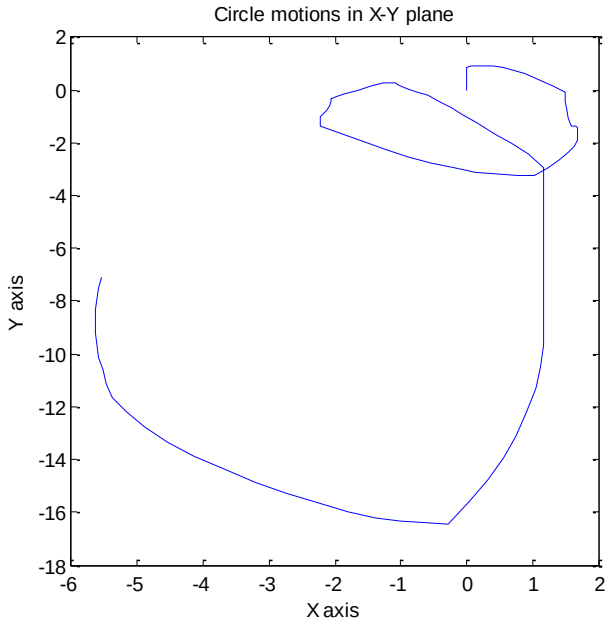


Figure 20: Same circular plotted only in X and Y coordinates

For this movement, two "circles" were created in a clockwise fashion. The first "circle" was recreated as an oval shape, and the second circle is incomplete. Also, the recreated path above shows the first circle smaller than expected. While moving the IMU, it was intended that both circles had roughly the same size.

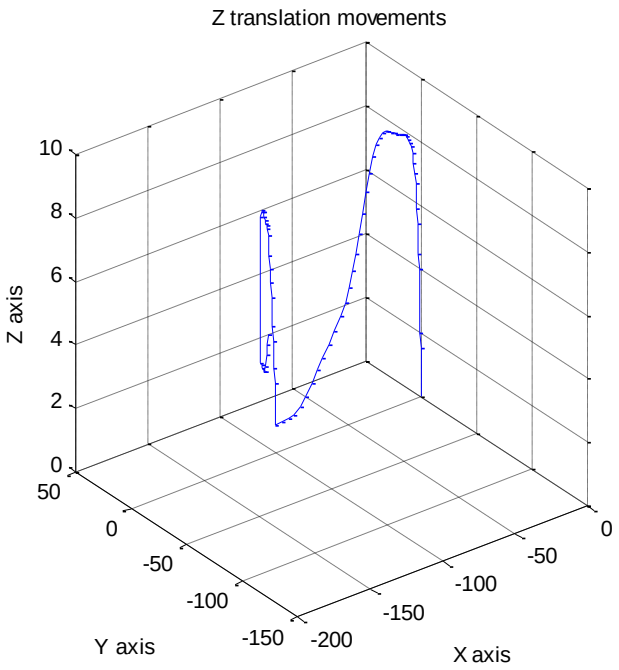


Figure 21: Up down motion

The path recreated above seems fairly accurate. The desired path starting from the point (0, 0, 0): straight up, then down and to the left; this was followed by an up-down motion. Upon further inspection, the Y-direction range shown above is actually 100cm, while the X-direction range is over 150cm. This was definitely not real path (at most, the Y-direction was 5cm and X-direction 15cm), which leads to believe the orientation was changing throughout the motion (as was the case). As described earlier, slight orientation changes cause the IMU/program to think it is moving due to gravity when it is actually not (it is simply just rotating). The general path outlined is fine, but the ranges are completely off.

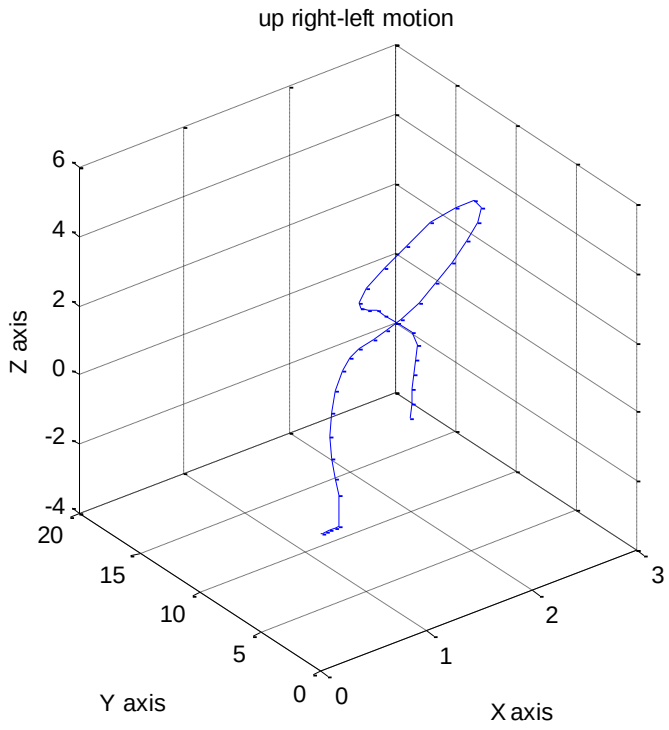


Figure 22: Up right-left motion

The intended path in Figure 22 was: directly up (Z direction), then right-left (staying in XY plane), and finally a negative-Z direction. It sort of looks like the path described, but errors make it look more like a loop motion.

The current method used to calculate position can be thought of as an open-loop system. Without feedback, errors cannot be compensated for (reduced) and these terms will subsequently be accumulated. Although it is true the "general" motion has been captured, these were for at most a 15 second span. If errors seem prevalent now, there's no chance such a tracking system will have any worth for operations that may take hours. Fortunately, there are methods which can be implemented to inertial tracking systems to greatly improved performance and handle slight errors. These methods will be briefly suggested in the recommendations chapter.

6 Conclusions and Recommendations

It is clear that although tracking is roughly achieved, much improvement can (and should) be made. Through this project, it is shown that simple "position from inertial" algorithms can achieve some (though limited) success in tracking the broad movements. Besides gravity compensation (mentioned in the previous chapter), improvements for systems based on purely inertial sensors may include: integration methods with less error, faster sampling rates, and implementing filtering methods.

For future considerations, accelerometer and gyroscope data can be incorporated together to provide improved results. Using accelerometer's response to Earth's gravitational field, we can determine tilt. Tilt is the angle displacement, which is identical to the angle calculated by integrating gyroscope data. These two measurements can be compared to offer more accurate angles. The angles, in turn, are used to calculate the rotation matrix which transforms the positions (in the moving frame) into positions with respect to a universal (or global) frame. Thus, implementing the above would yield a much better tracking system overall. A similar technique known as Kalman filtering can reduce the effect of accelerometer vibrational noise. In addition to these methods, integrating the inertial sensor system with another form of tracking system (e.g. optical system) is the ideal solution for tracking laparoscopic instruments (if using inertial sensors).

Although studying motion analysis for objective assessment purposes can ultimately lead to some form of standard in which the student has to pass, it does not demonstrate the actual surgical competence of the individual. There are other skills besides technical ones such as anatomical or protocol knowledge that also needs assessment. Consequently, tracking laparoscopic instruments should be purely used for objective assessment for manual skills.

In conclusion, this project has demonstrated the inertial sensors ability to track trajectories, which can ultimately be mounted on laparoscopic instruments to provide an objective assessment. It is a cost-effective way for upgrading the resolution for current tracking systems, where its small package will not impede surgical performance.

Appendix A: Amplification System

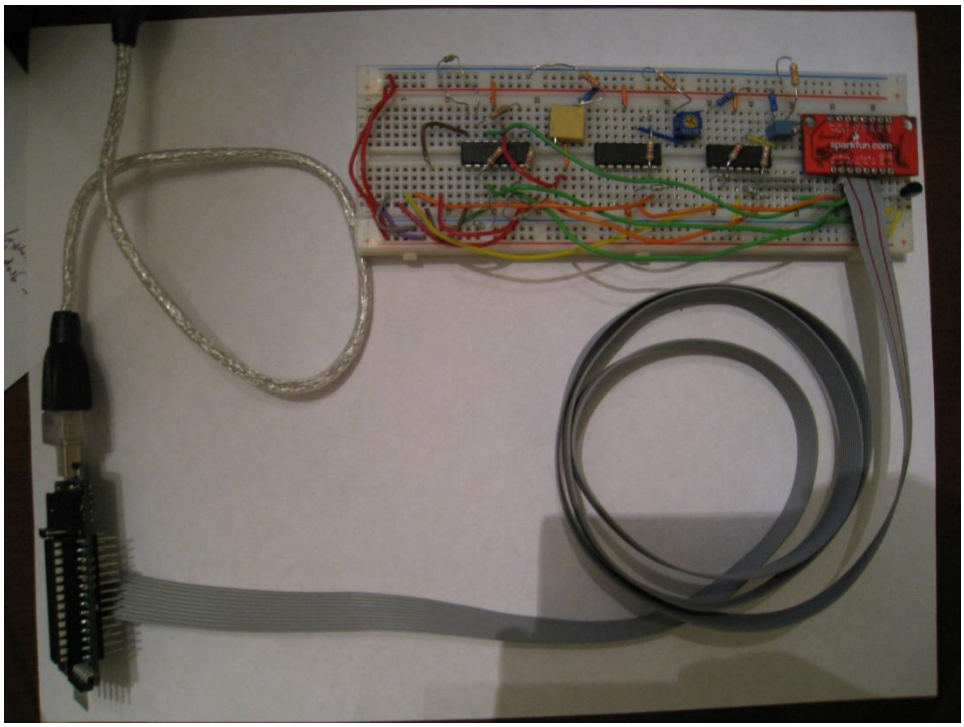
: 

Figure 23: Amplification system (top right of Figure) used to increase ADC performance

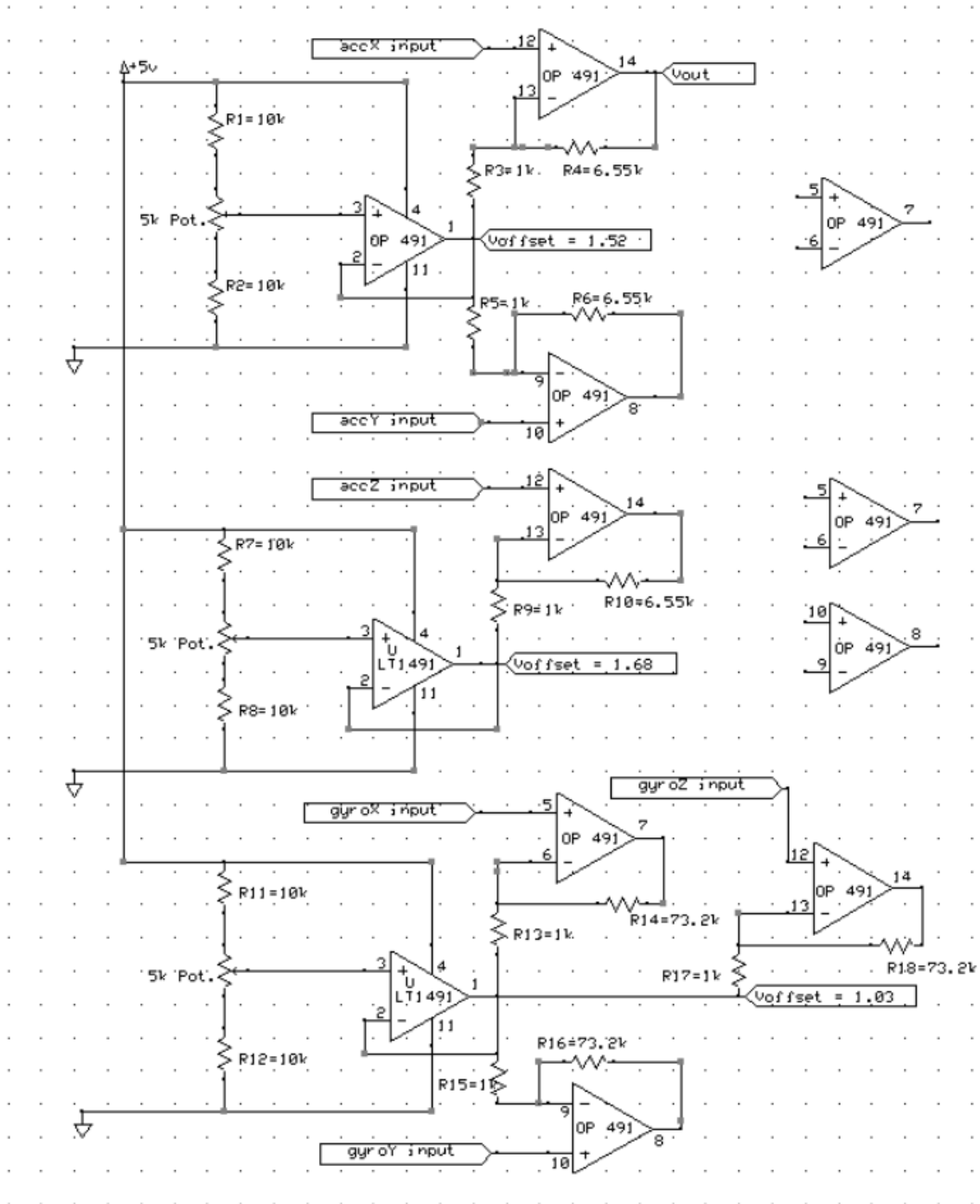


Figure 24: Amplify and offset system used with ADC module of reference voltage ±10V

- Output pins are 14, 8, and 7.
- Op-amps used were OP491, supplied by Dr. Patriciu.

Appendix B: Arduino Code

```
/*
 Inertial sensor tracking
 by Calvin Gan
 Language: Arduino
 Created: April 2, 2010
 */

/*** GLOBAL VARIABLES ***/
double st = 0; //start time
double dt = 0; //delta time; used to calculate integration intervals

//angular velocity samples from ADC (ints).
double sample_gyro_X;
double sample_gyro_Y;
double sample_gyro_Z;

//acceleration samples from ADC.
double sample_acc_X;
double sample_acc_Y;
double sample_acc_Z;

//counters for move_end_check() functions
int countx, county, countz;
int countgx, countgy, countgz;
//proportional values from accelerometer
double accX[2], accY[2], accZ[2]; //read from ADC (integers)
double velX[2], velY[2], velZ[2]; //first integration
double posX[2], posY[2], posZ[2]; //2nd integration

//proportional values from gyroscope, then converted to radians/s
double angVelX[2], angVelY[2], angVelZ[2];
double angleX[2],angleY[2],angleZ[2]; //first integration

//double sstateX, sstateY, sstateZ; //calibration values
double sstate_gyro_X, sstate_gyro_Y, sstate_gyro_Z;
double sstate_acc_X, sstate_acc_Y, sstate_acc_Z;

double R_base[3][3] = {{0}};

/*** FUNCTION PROTOTYPES ***/
void Calibrate(void);
void init_globalVar(void); //initializes global variables
void angularVelIntegrate(void);
void accelerationIntegrate(void);
void movementEndCheck();
//void movement_end_check_gyro();
void multiplyMatrices(double a[][3], double b[][3], double result[][3]);
void calculateRotation(double angleX, double angleY, double angleZ,double R_curr[][3]);
void matrixByVector(double a[], double result[]);

double convertToRadiansPerSec(double anglularVelADC);
double convertToMetersPerSecSq(double posADC);
```

```
double PosX=0,PosY=0,PosZ=0;
double AngleX=0,AngleY=0,AngleZ=0; //easier way to serial.print

/*** BEGIN SETUP AND LOOP ***/
void setup() {
  analogReference(EXTERNAL); //use external reference voltage for AD
  //Initialize the serial port
  Serial.begin(9600);
   //calibrate
  Calibrate();
  st= millis();
}

void loop() {
  dt = (millis()-st)/1000; //to calculate seconds
  st = millis(); //start time is the number of ms since the Arduino board began running the current program

  sample_gyro_X = convertToRadiansPerSec(analogRead(1));
  sample_gyro_Y = convertToRadiansPerSec(analogRead(0));
  sample_gyro_Z = convertToRadiansPerSec(analogRead(2));

  sample_acc_X = convertToMetersPerSecSq(analogRead(5));
  sample_acc_Y = convertToMetersPerSecSq(analogRead(4));
  sample_acc_Z = convertToMetersPerSecSq(analogRead(3));

   angularVelIntegrate();
   accelerationIntegrate();

  // Output values to serial port as an ASCII numeric string
  Serial.print(AngleX);
  Serial.print(",");
  Serial.print(AngleY);
  Serial.print(",");
  Serial.print(AngleZ);
  Serial.print(",");
  Serial.print(PosX);
  Serial.print(",");
  Serial.print(PosY);
  Serial.print(",");
  Serial.print(PosZ);
 // Serial.print(",");
  //Serial.print(dt);
  Serial.println();
  //Print a newline and carriage return in the end
 }

/*** CALIBRATE AND INIT GLOBALVAR ***/
void Calibrate(void) {
  init_globalVar();
  unsigned int count1;
  count1 = 0;
  do {
    double dt;
    //    sample_gyro_X = analogRead(1);
    //    sample_gyro_Y = analogRead(0); //x gyro corresponds to -y acc axis
```

```
    //    sample_gyro_Z = analogRead(2); //y gyro corresponds to -x acc axis
    //    sample_acc_X = analogRead(5);
    //    sample_acc_Y = analogRead(4);
    //    sample_acc_Z = analogRead(3);

    sstate_gyro_X = sstate_gyro_X + analogRead(1); //accumulate samples
    sstate_gyro_Y = sstate_gyro_Y + analogRead(0);
    sstate_gyro_Z = sstate_gyro_Z + analogRead(2);
    sstate_acc_X = sstate_acc_X + analogRead(5); //accumulate samples
    sstate_acc_Y = sstate_acc_Y + analogRead(4);
    sstate_acc_Z = sstate_acc_Z + analogRead(3);
    count1++;
   }
   while (count1 != 100); //100 samples

   //zero-state level accelerometer
   sstate_acc_X = sstate_acc_X / count1; //divide by 100
   sstate_acc_Y = sstate_acc_Y / count1;
   sstate_acc_Z = sstate_acc_Z / count1;

   //zero-state level gyroscope
   sstate_gyro_X = sstate_gyro_X / count1; //divide by 100
   sstate_gyro_Y = sstate_gyro_Y / count1;
   sstate_gyro_Z = sstate_gyro_Z / count1;

   // build the current rotation matrix
   calculateRotation(0.0, 0.0, 0.0, R_base);
}//calibrate

/*** INIT GLOBAL VARIABLES: ***
 * note: may be faster to initialize when first declared
 */
void init_globalVar() {
   sample_gyro_X = 0, sample_gyro_Y = 0,  sample_gyro_Z = 0;
   sample_acc_X = 0,  sample_acc_Y = 0,  sample_acc_Z = 0;
   //counters for move_end_check() functions
   countx = 0, county = 0, countz = 0;
   countgx = 0, countgy = 0, countgz = 0;
   //accelerometer values
   accX[0] = 0, accY[0] = 0, accZ[0] = 0;
   velX[0] = 0, velY[0] = 0, velZ[0] = 0;
   posX[0] = 0, posY[0] = 0, posZ[0] = 0;
   accX[1] = 0, accY[1] = 0, accZ[1] = 0;
   velX[1] = 0, velY[1] = 0, velZ[1] = 0;
   posX[1] = 0, posY[1] = 0, posZ[1] = 0;

   //gyroscope values
   angVelX[0] = 0, angVelY[0] = 0, angVelZ[0] = 0;
   angVelX[1] = 0, angVelY[1] = 0, angVelZ[1] = 0;
   angleX[0]=0, angleY[0]=0, angleZ[0]=0;
   angleX[1]=0, angleY[1]=0, angleZ[1]=0;

   //calibration values
   sstate_gyro_X = 0, sstate_gyro_Y = 0, sstate_gyro_Z = 0;
   sstate_acc_X = 0, sstate_acc_Y = 0, sstate_acc_Z = 0;
   return;
```

```
}//init


void angularVelIntegrate() {

 double dw =0; //discrimination window
 double dThetaX = 0, dThetaY = 0, dThetaZ = 0;
 double R_curr[3][3] = {{0}};
 double R_baseTemp[3][3] = {{0}};

 //memcpy(R_baseTemp, R_base, 8 * 9);
 int a=0,b=0;
 for (a=0;a<3;a++){
  for (b=0;b<3;b++){
   R_baseTemp[a][b]=R_base[a][b];
  }
 }

 //analogRead for Gyroscope: 0 -> Y, 1 -> X, 2 -> Z
 angVelX[1] = -sample_gyro_Y + convertToRadiansPerSec(sstate_gyro_Y); //to acheive RH frame
 angVelY[1] = sample_gyro_X - convertToRadiansPerSec(sstate_gyro_X);
 angVelZ[1] = sample_gyro_Z - convertToRadiansPerSec(sstate_gyro_Z);
 //angular velocity now in SI units.

 //apply discrimination window
 dw = 0.3;
 if ((angVelX[1] <= dw) && (angVelX[1] >= -dw)) {
  angVelX[1] = 0;
 }
 if ((angVelY[1] <= dw) && (angVelY[1] >= -dw)) {
  angVelY[1] = 0;
 }
 if ((angVelZ[1] <= dw) && (angVelZ[1] >= -dw)) {
  angVelZ[1] = 0;
 }

 //integrate
 dThetaX = ((angVelX[1] + angVelX[0]) * 0.5) * dt;
 dThetaY = ((angVelY[1] + angVelY[0]) * 0.5) * dt;
 dThetaZ = ((angVelZ[1] + angVelZ[0]) * 0.5) * dt;

 angleX[1]=angleX[0]+dThetaX;
 angleY[1]=angleY[0]+dThetaY;
 angleZ[1]=angleZ[0]+dThetaZ;

 //current velocity sent to previous velocity
 angVelX[0] = angVelX[1]; //radians/s
 angVelY[0] = angVelY[1];
 angVelZ[0] = angVelZ[1];

 angleX[0] = angleX[1];
 angleY[0] = angleY[1];
 angleZ[0] = angleZ[1];

 calculateRotation(dThetaX, dThetaY, dThetaZ, R_curr);
 multiplyMatrices(R_baseTemp, R_curr, R_base);
```

```
  AngleX = degrees(angleX[1]);
  AngleY = degrees(angleY[1]);
  AngleZ = degrees(angleZ[1]);

}//angularVelIntegrate


void accelerationIntegrate() {

  double dw=0; //discrimination window
  double dwMax = 5; //max speed of 5 m/s^2
  double accVector[3]={0,0,0};
  double accAfterTransform[3] = {0,0,0};
  double dPosX=0, dPosY=0, dPosZ=0;

  double zeroLevelX = convertToMetersPerSecSq(sstate_acc_X);
  double zeroLevelY = convertToMetersPerSecSq(sstate_acc_Y);
  double zeroLevelZ = convertToMetersPerSecSq(sstate_acc_Z);
  //subtract calibration level (obtain positive/negative velocity)
  accX[1] = sample_acc_X - zeroLevelX;
  accY[1] = sample_acc_Y - zeroLevelY;
  accZ[1] = sample_acc_Z - zeroLevelZ;

  //discrimination window: used for no-movement condition
  //related to accelerometer noise
  dw = 0.3;
  if ((accX[1] <= dw) && (accX[1] >= -dw)) {
    accX[1] = 0;
  }
  if ((accY[1] <= dw) && (accY[1] >= -dw)) {
    accY[1] = 0;
  }
  if ((accZ[1] <= dw) && (accZ[1] >= -dw)) {
    accZ[1] = 0;
  }

  //if acceleration is very fast, its likely due to an unwanted gravity component, so ignore it
  if ((accX[1]>=dwMax)||(accX[1]<=-dwMax)) {
    accX[1]=0;
  }
  if ((accY[1]>=dwMax)||(accY[1]<=-dwMax)) {
    accY[1]=0;
  }
  if ((accZ[1]>=dwMax)||(accZ[1]<=-dwMax)) {
    accZ[1]=0;
  }

  //transform acceleration from moving frame to base frame
  //R_base*acc:: 3x3 matrix * vector
  accVector[0] = accX[1];
  accVector[1] = accY[1];
  accVector[2] = accZ[1];

  //transform acceleration to base frame by multiplying by R_base
  matrixByVector(accVector, accAfterTransform);
```

```
  //do subtraction
  accX[1] = accAfterTransform[0];  //-gVector[0];
  accY[1] = accAfterTransform[1];  //-gVector[1];
  accZ[1] = accAfterTransform[2];  //-gVector[2];

  /********** integrate ********/
  velX[1] = velX[0] + ((accX[1] + accX[0]) * 0.5) * dt; //velocity (m/s)
  velY[1] = velY[0] + ((accY[1] + accY[0]) * 0.5) * dt;
  velZ[1] = velZ[0] + ((accZ[1] + accZ[0]) * 0.5) * dt;

  dPosX = ((velX[1] + velX[0]) * 0.5) * dt;//deltaPosition in meters
  dPosY = ((velY[1] + velY[0]) * 0.5) * dt;
  dPosZ = ((velZ[1] + velZ[0]) * 0.5) * dt;

  posX[1] = posX[0] + dPosX; //meters
  posY[1] = posY[0] + dPosY;
  posZ[1] = posZ[0] + dPosZ;

  movementEndCheck();

  //current value must be sent to previous value
  accX[0] = accX[1];
  accY[0] = accY[1];
  accZ[0] = accZ[1];

  velX[0] = velX[1];
  velY[0] = velY[1];
  velZ[0] = velZ[1];

  //in meters
  posX[0] = posX[1];
  posY[0] = posY[1];
  posZ[0] = posZ[1];

  PosX = posY[1]*100;
  PosY = posX[1]*100;
  PosZ = posZ[1]*100;

}//accIntegrate

void movementEndCheck(){

  if (accX[1] == 0) {
    countx++;
  }//we count the number of velocity samples that equals zero
  else {
    countx = 0;
  }
  if (countx >= 4) { //if this number exceeds 25, we can assume that velocity is zero
    velX[1] = 0;
    velX[0] = 0;
  }

  if (accY[1] == 0) {
    county++;
```

```
  }//we count the number of velocity samples that equals zero
  else {
    county = 0;
  }
  if (county >= 4) {
    velY[1] = 0;
    velY[0] = 0;
  }

  if (accZ[1] == 0) {
    countz++;
  }//we count the number of velocity samples that equals zero
  else {
    countz = 0;
  }
  if (countz >= 4) {
    velZ[1] = 0;
    velZ[0] = 0;
  }
  return;

}//movementEndCheck

void multiplyMatrices(double a[][3], double b[][3], double result[][3]) {
  double sum;
  int i, j, k;
  for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++) {
      sum = 0;
      for (k = 0; k < 3; k++)
        sum += a[i][k] * b[k][j];
      result[i][j] = sum;
    }
}
void calculateRotation(double angleX, double angleY, double angleZ,double R_curr[][3]) {

  //Rx(angleX)
 // if (angleX!=0 && angleY!=0 && angleZ!=0){
  double Rx[3][3] = {{ 1, 0, 0} , {0, cos(angleX), -sin(angleX)}, {0,sin(angleX), cos(angleX)}};
  double Ry[3][3] = {{cos(angleY),0, sin(angleY)},{0, 1, 0},{-sin(angleY),0,cos(angleY)}};
  double Rz[3][3] = {{cos(angleZ), -sin(angleZ),0},{sin(angleZ),cos(angleZ),0},{0, 0, 1}};
  double Rxy[3][3] = {{0}};
  multiplyMatrices(Rx, Ry, Rxy);
  multiplyMatrices(Rxy, Rz, R_curr);
  //}
}
//multiply matrix (3x3) by a vector (3x1)
void matrixByVector(double a[], double result[]) {
  int i = 0, j = 0;
  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      result[i] = result[i] + R_base[i][j] * a[j];
    }
  }
}
```

```
double convertToRadiansPerSec(double anglularVelADC) {
  //angularVel is ADC reading (proportional value)
  //degrees = (angle * Vref / 1023 - VzeroRate) / Sensitivity
  //already subtract zero rate
  int vref = 3.3; //ADC module ref. voltage
  double sensGyro = 0.00333; //* 2.7; //gyro sensitivity in V/degree/s
  double degrees1 = ((anglularVelADC * vref / 1023) / sensGyro);
  return radians(degrees1);
}
double convertToMetersPerSecSq(double posADC) {
  int vref = 3.3; //20?
  double sensAcc = 0.33;// * 3.92; //in v/g
  double accInMeters = ((posADC * vref / 1023) / sensAcc); //will get acceleration in g's
  return accInMeters * 9.8;
}
```

Appendix C: Tables for Graphs

The graph created in Figure 16 and 17 in the results section were generated from the following table:

Please note that while graphing the following values, the Ax and Az values were negated (created negative). This was for visual purposes only; the initial orientation of the IMU was such that an "up" (z-direction) movement would result in a negative z-direction, and a "right" (x-direction) movement would result in a negative x-direction. Therefore, for the graphs to mimic the intended motions (i.e. a "right" motion displaying as "right" on the graph), the Ax and Ay values should be negative the values shown below.

Table 5: Table used to create Figure 16 and 17.

Gx	Gy	Gz	Ax	Ay	Az
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	-0.01	0
0	0	0	-0.03	-0.02	0
0	0	0	-0.1	0.01	0
0	0	0	-0.21	0.07	0
0	0	0	-0.31	0.13	0
0	0	0	-0.42	0.19	0
0	0	0	-0.42	0.19	0
0	0	0	-0.42	0.19	0
0	0	0	-0.42	0.19	0
0	0	0	-0.42	0.19	0
0	0	0	-0.42	0.19	0
0	0	0	-0.42	0.22	0
0	0	0	-0.42	0.38	0
0	0	0	-0.4	0.73	0
0	0	0	-0.38	1.26	0
0	0	0	-0.37	1.9	0
0	0	0	-0.36	2.54	0
0	0	0	-0.32	3.1	0
0	0	0	-0.27	3.58	0
0	0	0	-0.24	3.99	0
0	0	0	-0.23	4.35	0
0	0	0	-0.26	4.68	0

0	0	0	-0.31	4.95	0
0	0	0	-0.35	5.12	0
0	0	0	-0.36	5.19	0
0	0	0	-0.37	5.22	0
0	0	0	-0.37	5.26	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.37	5.29	0
0	0	0	-0.34	5.29	0
0	0	0	-0.21	5.29	0
0	0	0	0.03	5.29	0
0	0	0	0.38	5.29	0
0	0	0	0.82	5.29	0
0	0	0	1.31	5.29	0
0	0	0	1.79	5.29	0
0	0	0	2.28	5.29	0
0	0	0	2.8	5.29	0
0	0	0	3.3	5.29	0
0	0	0	3.78	5.29	0
0	0	0	4.22	5.29	0
0	0	0	4.58	5.29	0
0	0	0	4.85	5.29	0
0	0	0	5.01	5.29	0
0	0	0	5.07	5.29	0
0	0	0	5.07	5.29	0
0	0	0	5.06	5.29	0
0	0	0	5.05	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0

0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.03	5.29	0
0	0	0	5.02	5.28	0
0	0	0	4.98	5.21	0
0	0	0	4.92	5.04	0
0	0	0	4.87	4.74	0
0	0	0	4.81	4.36	0
0	0	0	4.8	3.94	0
0	0	0	4.76	3.51	0
0	0	0	4.69	3.07	0
0	0	0	4.61	2.65	0
0	0	0	4.53	2.25	0
0	0	0	4.46	1.84	0
0	0	0	4.41	1.45	0
0	0	0	4.37	1.07	0
0	0	0	4.34	1.07	0
0	0	0	4.3	1.09	0
0	0	0	4.3	1.18	0
0	0	0	4.3	1.36	0
0	0	0	4.3	1.6	0
0	0	0	4.3	1.89	0
0	0	0	4.3	2.16	0
0	0	0	4.3	2.44	0
0	0	0	4.3	2.44	0
0	0	0	4.3	2.44	0
0	0	0	4.3	2.44	0
0	0	0	4.3	2.44	0
0	0	0	4.3	2.44	0
0	0	0	4.29	2.44	0
0	0	0	4.23	2.43	0
0	0	0	4.1	2.39	0
0	0	0	3.85	2.32	0
0	0	0	3.5	2.24	0
0	0	0	3.07	2.17	0
0	0	0	2.6	2.09	0

0	0	0	2.04	2.09	0
0	0	0	1.45	2.09	0
0	0	0	0.86	2.09	0
0	0	0	0.27	2.09	0
0	0	0	-0.31	2.09	0
0	0	0	-0.88	2.09	0
0	0	0	-1.42	2.09	0
0	0	0	-1.92	2.09	0
0	0	0	-2.3	2.09	0
0	0	0	-2.54	2.09	0
0	0	0	-2.68	2.09	0
0	0	0	-2.8	2.09	0
0	0	0	-2.91	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.03	2.09	0
0	0	0	-3.04	2.13	0

0	0	0	-3.09	2.26	0
0	0	0	-3.17	2.54	0
0	0	-0.32	-3.27	2.91	0
0	0	-0.65	-3.39	3.39	0
0	0	-1.04	-3.55	3.97	0
0	0	-1.8	-3.73	4.61	0
0	0	-2.52	-3.91	5.27	0
0	0	-3.21	-4.09	5.88	0
0	0	-3.89	-4.26	6.4	0
0	0	-4.23	-4.42	6.81	0
0	0	-4.23	-4.57	7.05	0
0	0	-4.23	-4.71	7.14	0
0	0	-4.23	-4.84	7.11	0
0	0	-4.23	-4.97	7.04	0
0	0	-4.55	-5.1	6.96	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0

0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0
0	0	-4.86	-5.23	6.89	0

REFERENCES

[1] Rattner DW, Ferguson C, Warshaw AL. Factors associated with successful laparoscopic cholecystectomy for acute cholecystitis. Ann Surg. 1993 Mar;217(3):233–236

[2] Learn About Laparoscopy. Previous experience of open surgery does not affect the outcome of developing laparoscopic skill. [Online] Available: http://laparoscopytraining.com [Accessed: Oct 8, 2009]

[3] Ikuta, K., Kato, T., Ooe, H., & Shinohara, K. (2008). Surgery recorder system acquiring position/force information of surgical forceps. Paper presented at the *Automation Congress, 2008. WAC 2008. World,* 1-6.

[4] Chmarra, M.K., Gimbergen, C. A. and Dankelman, J.(2007). Systems for tracking minimally invasive surgical instruments. Minimally Invasive Therapy and Allied Technologies, 2007, 16:6,320 – 340

[5] E. Foxlin, M. Harrington, and Y. Altshuler. Miniature 6- DOF Inertial System for Tracking HMDs. In Proc. SPIE Helmet and Head-Mounted Displays III, vol. 3362, Orlando, April, 1998.

[6] Gregory Baratoff and Scott Blanksteen, “Tracking Devices” [Online] Available: http://www.hitl.washington.edu/scivw/EVE/I.D.1.b.TrackingDevices.html [Accessed: April 7, 2010]

[7] Tatar, F.; Mollinger, J.; Bossche, A.; , "Ultrasound system for measuring position and orientation of laparoscopic surgery tools," *Sensors, 2003. Proceedings of IEEE* , vol.2, no., pp. 987- 990 Vol.2, 22-24 Oct. 2003

[8] Rosen, J.; Brown, J.D.; Chang, L.; Barreca, M.; Sinanan, M.; Hannaford, B.; , "The BlueDRAGON - a system for measuring the kinematics and dynamics of minimally invasive surgical tools in-vivo," *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on* , vol.2, no., pp. 1876- 1881 vol.2, 2002

[9] Ji-Hwan Kim, Nguyen Duc Thang, Hyun Sang Suh, Rasheed, T. and Tae-Seong Kim. Forearm Motion Tracking with Estimating Joint Angles from Inertial Sensor Signals 2009

[10] Kim, A., Golnaraghi, M. F. Initial calibration of an inertial measurement unit using an optical position tracking system 2004

[11] Lamata, P., Gomez, E. J., Hernandez, F. L., Pastor, A. O., Sanchez-Margallo, F. M. and del Pozo Guerrero, F. Understanding Perceptual Boundaries in Laparoscopic Surgery 2008

[12] Speidel, S., Delles, M., Gutt, C. and Dillmann, R. Tracking of instruments in minimally invasive surgery for surgical skill analysis 2006

Vitae

Name:	Calvin Gan
Date of Birth:	January 2, 1987
Secondary Education:	Clarkson Secondary School (2005)
Post Secondary Education:	Electrical & Biomedical Engineering, McMaster University (2010)