4-18-2017

Sensory Interfaces: Intelligent Surgical Box Trainer

Krystal York
Western Michigan University, krystalyork1696@gmaill.com

Sensory Interfaces for an Intelligent Surgical Box Trainer

Authors:

Joshua Clemens

Sarah Seng

Krystal York

Advisor and Sponsor:

Dr. Janos Grantner

Electrical and Computer Engineering 4820

April 25, 2017

Table of Contents

Abstract

The Fundamentals of Laparoscopic Surgery (FLS) box trainer currently offers medical students the opportunity to practice their dexterity with laparoscopic surgical tools and accustom themselves to the disconnect between their actions and their visible frame of reference. However, at this stage the box trainer offers few ways to test, practice, or improve upon important factors in laparoscopic surgery including disturbance of and pressure applied to patient tissue. The purpose of this project was to add sensory interfaces to the FLS box trainer to offer more concrete feedback for medical students practicing the required tasks, consolidate the information gathered by the sensors, and export it to the box trainer's computer for the study and use of the medical student. The system developed placed a force sensor at the handle of the grasper tool which measures the force applied to the tool by the student, an accelerometer at the student's wrists to measure their orientations while using the tools, and a timer in the software to record the time it took the students to complete each task. The system was implemented using an STM32F4 Discovery Board microcontroller with an ARM Cortex-M4 processor. The communication between the microcontroller and the PC was facilitated using ST software and the Discovery Board's embedded debugger and programmer STLink. Data from the sensors is saved to the box trainer PC using this connection. All specifications have been addressed and completed. The result is an extension to the box trainer which properly reads applied force, wrist orientation, and elapsed time of box trainer tests. Each sensor was properly calibrated. The program was compiled and included in this document. Medical students can now better see and understand how they are doing in real time while completing the required tasks.

1.0 Introduction

1.1 Background

Laparoscopic surgery is a form of surgery that is minimally invasive and has proven to have many distinct benefits. Laparoscopic surgery is performed using long and skinny tools that are inserted into small incisions in the abdomen wall. Some of the main advantages of this technique include smaller scars, shorter recovery time, shorter hospital stays for patients [1]. However, laparoscopic surgery requires skilled surgeons in order to effectively perform minimally invasive procedures. Specific psychomotor skills are necessary to perform the surgery because the surgeons are using tools to operate on tissue they have to see with a camera [2]. As detailed by Westebring, "the indirect vision through an endoscope (camera), and the indirect manipulation of tissue are the main causes of perception problems", which makes laparoscopic surgery a difficult procedure to master [3].

These disadvantages created a need for there to be a system where surgical students and practicing surgeons can easily and safely improve their laparoscopic surgery skills. The Fundamentals of Laparoscopic Surgery (FLS) program was created in 2004 to educate and test surgical residents and practicing surgeons [4]. Part of the examination process for the FLS program includes a physical test using an intelligent surgical box trainer. The box trainer essentially simulates laparoscopic surgery by offering various tasks that testers have to complete. All tasks are completed using the same tools that are used during surgery. The box trainer helps to build necessary psychomotor skills without practicing on a human individual.

The FLS program now certifies surgeons and "since the inception of FLS, more than 9,000 surgical residents, fellows, and practicing physicians have successfully completed the FLS program [4]." This provides insight into the importance of this technology and shows that this growing technology could benefit a large amount of people if further improvements to the system were made.

1.2 Need Statement

Even though intelligent surgical box trainers have proven to be effective tools to improve laparoscopic surgical skills, there are many shortcomings that need to analyzed. If there was a simpler way to analyze performance on an exam, testers could refer to their results to increase their abilities. Also, if the process was easier to analyze, the tests could be practiced without the need for a practicing surgeon to oversee the process. This would save money and would help surgical residents and practicing surgeons harness their laparoscopic skills anytime they wish.

A system of sensors needs to be implemented in the FLS box trainer system so information regarding applied force of the grabber tool, wrist movement, and test duration can interface with a microcontroller. The microcontroller needs to upload the performance feedback to a computer so it can be analyzed by test takers.

2.0 System Overview

The block diagram in *Figure 1* displays the system that this group was in charge of designing to add to the FLS box trainer to provide better feedback for the practicing surgeons.

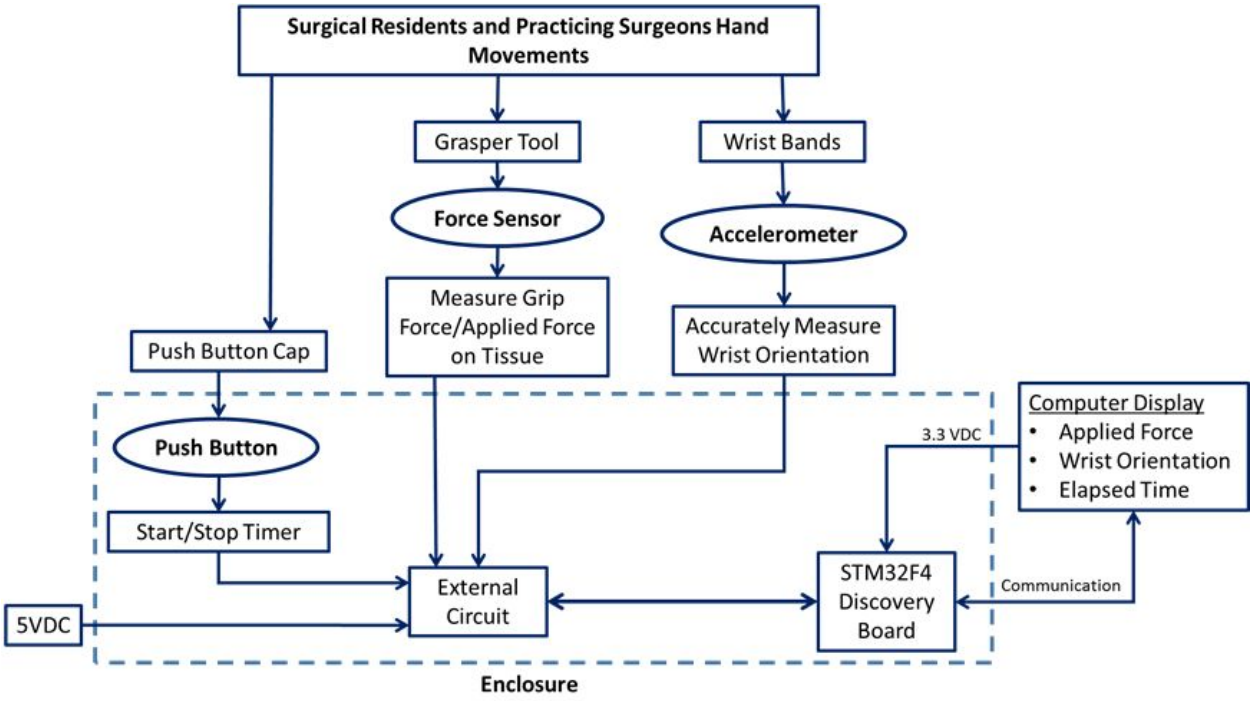

Figure 1: System Block Diagram

2.1 Requirement Specifications

In order to meet the needs of the WMU ECE Department and the WMU School of Medicine, the design for the improved FLS Trainer must include both hardware and software components. The scope of the hardware side of the design includes three sets of sensors that should measure the student performance parameters asked for in the need statement; a microcontroller that will intercept signals from these sensors, parse them, and log them in a data file; and interfacing components. The scope of the software side of the design includes programming the microcontroller to accept the three sensor signals, organize the data collected from those inputs, and store this data in a comprehensive format that will be saved on the FLS Trainer desktop.

The following specifications are organized into three basic categories. The first analyzes the needs and specifications for the sensors measuring the wrist orientations of the student, the force the student applies to the grasper, and the time elapsed during the FLS Training test. The second specifies the requirements for the microcontroller and software. The third captures any other miscellaneous conditions that must be met by the system design. Furthermore,

specifications were designated as being requirements, goals, or preferences with the letters [R], [G], and [P] respectively.

2.1.1 Sensory Interface Specifications

a. *Sensor able to accurately measure pressure from 0.1 to 4 PSI [R]* As it is discussed in the Literature Survey, human tissue is at risk of damage when more than 2.2 pounds per square inch of pressure is applied for a prolonged period of time. The pressure applied by the grasper tips should not exceed this number in a successful trial on the FLS Trainer. This sensory interface specifications would be considered a requirement because of the need for students to know how much pressure they are using.
b. *Sensor able to accurately measure a radial orientation range of 210º [R]* The FLS Trainer system requires a sensor that can accurately measure the orientation of a student's wrist throughout the test in order to ensure that the student's performance is ergonomically sound. As will be discussed in the Literature section, the maximum possible angular deviation of the wrist and elbow combined amounts to around 200º. This sensory interface specifications would be considered a requirement because of the need for students to know their wrist orientation.
c. *Sensor able to mark the beginning and ending of an FLS Trainer test [R]* The design is required to include a means by which to measure the time taken to complete the FLS Trainer test. This can be accomplished with a simple pushbutton that the student can tap at the start and conclusion of their test. The sensory interface specifications would be considered a requirement because of the need for students to know how long it took them to complete the test.
d. *All sensors must weigh less than 1 oz [R]* The FLS Trainer is meant to simulate the real experiences and challenges of performing laparoscopic surgery, so the design should not impede the student's movements in any way. For this reason, the sensors placed on the student's wrist and graspers must be as small as possible both in terms of volume and weight. This specification would be considered a requirement because the ability for the students to be tested without distraction is an important aspect of this design.
e. *Discernable analog or digital signals from sensors to microcontroller [R]* The sensors must generate signals that the microcontroller is capable of reading and storing. This is considered a requirement because of the need for the sensors to interact with the microcontroller and store data.

2.1.2 Processing and Logging Specifications

a. *Microcontroller system [R]: STM32F4 Discovery Board [P]* The sponsor for this project indicated a preference for the STM32F4 Discovery Board as the microcontroller used in the design. The STM32F4 is more than capable of the required tasks. The use of a microcontroller would be considered a requirement because of the need to control and send signals to the sensors. The use of the STM32F4 Discovery Board would be considered a preference since other microcontrollers could be used. This was a preference given by the project sponsor.
    i. *5 VDC power source [R]*

This microcontroller is powered from 5 VDC through a USB port. The local desktop computer of the FLS Trainer can supply this power. This is considered a requirement since the microcontroller being used needs 5 VDC to power it.

ii. *C programming [R]*
This software interface of the STM32F4 allows for programming the microcontroller directly using C language. This is considered a preference since the microcontroller being used is programmable with either C or assembly language.

b. *Serial connection required from microcontroller to PC [R]*
The sponsor also requested that the data transfer from the STM32F4 to the FLS Trainer desktop occur through a serial connection. Because there is a need for the medical students to see their results in real-time and for the microcontroller to have self-testing capabilities, the collection of sensory data to send it to a computer is considered a requirement.

c. *Data stored in accessible format [R] (excel file [P])*
The overarching purpose of this design is to provide additional feedback from the FLS Training tests to students and evaluators. This requires that these parties be able to access the gathered information in a clear and comprehensive manner. The collection of sensory data to send to a computer is considered a requirement, whereas the ability to display this data in an Excel file would be considered a preference since there could be other means of displaying the data.

2.1.3 System Specifications

a. *The microcontroller and other auxiliary electronics should be contained in a nonconductive enclosure [R]*
As this design is intended to serve supportive rather than primary functions, the physical components should be as out-of-the-way as is feasible. Containing all the hardware (besides the sensors, which must necessarily interact with the user) in a single enclosure accomplishes this. Specifying that the enclosure is nonconductive reduces the likelihood of interference or electrical damage to the equipment. Because the nonconductive enclosure is for safety purposes, this would be considered a requirement.

b. *The enclosure should be mounted on the FLS Trainer cart [R]* for safety and convenience. Because this specification is for safety purposes, this would be considered a requirement.

c. *The enclosure should weigh less than 1 lb. [G]* for efficiency and to minimize the device footprint. Because the weight of the enclosure does not necessarily affect the performance it would be considered a goal.

d. *Components should ideally require 3.3 VDC or 5 VDC power [G]*, which can be supplied using wall outlet converters. If this is not feasible, additional power converters/sources will need to be added to the enclosure. This is a goal set by the project advisor.

3.0 Hardware

3.1 Detailed Design Concept

Figure 2 illustrates the high level design concept for the box trainer sensory interfaces. This block diagram shows how the system interacts with the box trainer. As students or other users of the FLS Trainer system begin the test, their movements excites sensors integrated into the system. The signals generated by these sensors are processed by the Discovery Board microcontroller and stored in the Discovery Board memory. The values in these memory locations are viewable and stored in a log file with a program on the local PC for the user's review.

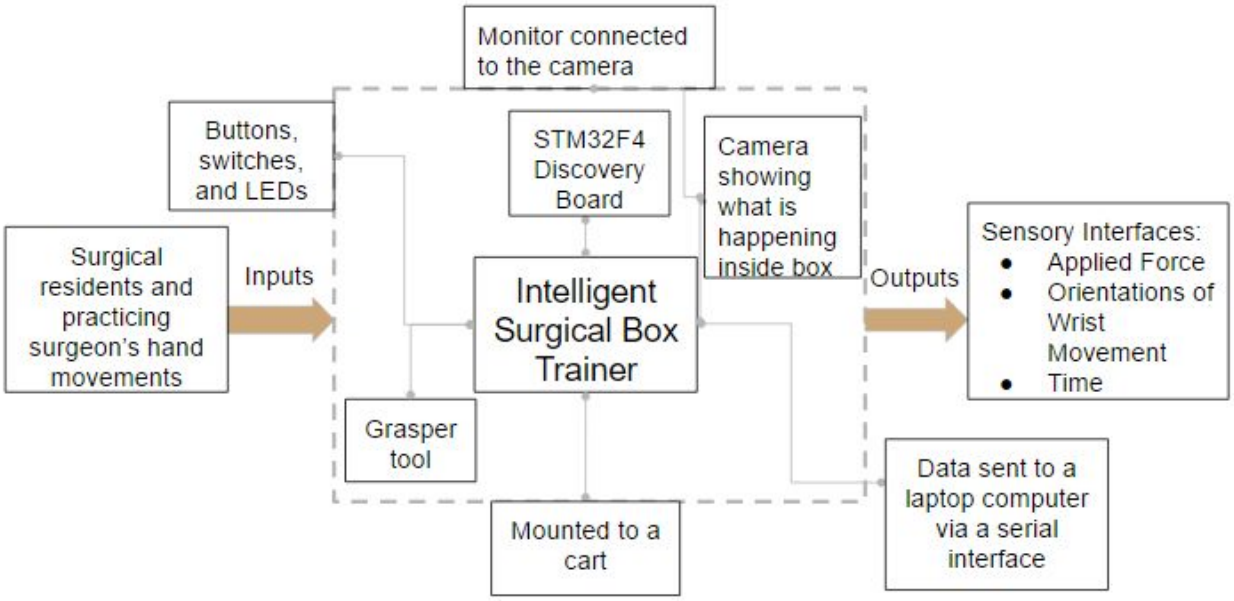

Figure 2: High Level Block Diagram

The hardware portion of the design consists of a microcontroller, a USB A to mini-B cable, a USB to serial cable, and three sensors (with their hardwire connections to the microcontroller). The software portion is a program written in C with IAR Embedded Workbench and extended ST libraries, which interfaces with the Discovery Board.

3.2 System Design Analysis

3.2.1 Discovery Board

The design uses an STM32F4 Discovery Board microcontroller, as specified by the sponsor. Each of the sensors discussed above is hardwired to the GPIO ports of the microcontroller using a UL certified 4-pair CAT5e 24-AWG signal cable. The force sensors outputs are connected to pins PB6 and PB7 of the Discovery Board. The accelerometer outputs

are connected to pins PC0, PC1, PC2, PC3, PC4, and PC5. The push button output is connected to pin PD0. The force sensors' supply voltage is connected to a 5 VDC wall adapter. The accelerometer and push button are powered by 3.3 VDC. A system ground connects all these sensor ground pins with the Discovery Board ground and the power supply grounds. The Discovery Board is powered from the local PC using a USB-A to mini-B cable. This cable also enables communication between the Discovery Board and the PC using the board's ST Link. The information gathered from the sensors is tracked and logged using STMStudio.

3.2.2 Force Sensors

The force sensor selected at the recommendation of the sponsor is the SingleTact tactile capacitive force sensor that can measure up to 22 pounds of force on an 8 millimeter sensor, amounting to a 282.37 psi capacity [5]. This force sensor is placed at the handle of each grasper tool. More about of how the grasper tool will interact with the force sensor will be discussed in section 3.2.7. Each sensor requires four conductors for power and signal transfer. The device generates an analog signal between 0 and 2 V in response to pressure and accepts supply voltages between 3.7 and 12 V [5]. This analog signal runs to a precalibrated I^2C interface board, which converts the 0-2V analog signal to a 0x000 to 0xFF digital output. The interface board connects the sensor to power and output SCL and SDA signals with force information to the Discovery Board. *Table 1* contains the pinout information for the sensor. *Figure 3* shows the hardware configuration for a force sensor. The software configuration will be detailed in section 5.0.

Table 1: Pressure Sensor Pinout

Pin Number	Connection
1	Vcc
2	Analog Output
3	I2C Interface (SCL)
4	Reserved
5	Reserved
6	I2C Interface (SDA)
7	Frame Sync
8	Ground

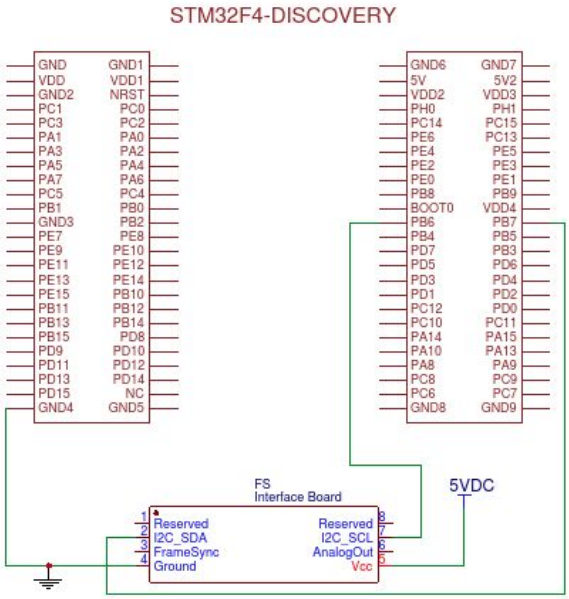

Figure 3: Hardware Configuration for a Force Sensor

3.2.3 Accelerometers

A three axis accelerometer is used to determine the orientation of the student's wrist. The design attaches an accelerometer to a wristband that will be worn during the FLS Trainer exercise. The wristband must be situated on the student's arm in such a way that the accelerometer z-axis (labeled on the chip) is pointed upward when the student's wrist is relaxed. In this way, the angular orientation of the students' wrists are standardized and may be evaluated against a standard set by the instructor.

The accelerometer selected based on provided specifications is an ADXL335 accelerometer with solderable breakout board. This sensor is sewn into a wristband, and the wristband is marked clearly with indications for how it should be situated on the user's wrist. The accelerometer requires five conductors for signal and power transfer. It can measure acceleration with a maximum full-scale range of ±3g, accepts power supply voltages ranging from 1.8V to 3.6V, and generates three analog output signals. *Table 2* contains the pinout information for this sensor. This design use the ground pins, the supply voltage pins, and the three output pins.

3.2.4 Push Button

The push button is used to start and stop the testing procedure. The main purpose of this normally open switch is to operate a timer that monitors the duration of the test while initializing the collection of data from the other two sensors. This specific push button was selected because it is very similar to the push buttons located on the discovery board and is easy to integrate in a circuit built on a breadboard.

Table 2: Accelerometer Pinout

Pin Number	Connection
1	No connect
2	Self-Test
3	Common Ground
4	No connect
5	Common Ground
6	Common Ground
7	Common Ground
8	Z Channel Output
9	No connect
10	Y Channel Output
11	No connect
12	X Channel Output
13	No connect
14	Supply Voltage
15	Supply Voltage
16	No connect

The biggest issue with the push button is that it needs to be pressed by the user with the grasper tool, inside the box trainer. A push button cap was designed to make this an easier task for the user, since the push button is less than a quarter inch in diameter. Using AutoCAD's 3D modeling feature, a push button cap was designed to have a inverse dome face on the top of the cap. This will make it easier for users to push the button, because wherever they press the cap, the grasper tool will slide to the center of the cap and the push button will be pressed. The 3D printer in the ECE Senior Design Lab was used to print the push button caps used for the project. The push button cap is mounted on the outside of the non-conductive enclosure the will house the circuit inside the box trainer.

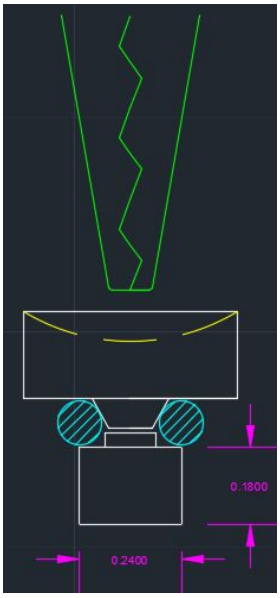

Figure 4: Push Button

3.2.5 Non-conductive enclosure

A specification for the project is that all of the electronics and microcontroller must fit inside the box trainer inside a non-conductive enclosure. Since specific dimensions are needed for the enclosure and a 3D printer is easily accessible, the non-conductive enclosure was modeled in AutoCAD and printed using a 3D printer. The 3D printer available in the ECE Senior Design Lab prints using ABS and PLA plastic. Both of these plastics can be specially made to be conductive, but in most cases, it serves as an insulator that is used to support electronics. The enclosure consists of two halves of a plastic shell that will serve as the non-conductive enclosure. *Figure 5* displays the 3D model made in AutoCAD of the non-conductive enclosure. The push button cap is located on the outside of the enclosure, but all other electronics, except the force sensor and accelerometers, will be located inside this enclosure.

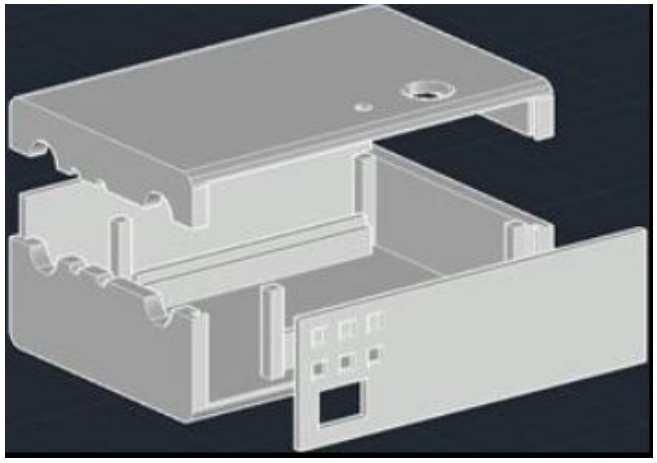

Figure 5: 3D printed Enclosure

3.2.6 Grasper Tools

A capacitive force sensor delivers the most robust and reliable performance for the purposes of this design. Because placing the force sensor on the tip of the graspers interferes with the user's performance, a capacitive force sensor is, instead, installed at the handle of the grasper. The team used another force sensor temporarily placed at the the tip of the grasper to determine the relationship between force applied at the handle of the grasper and the force of the grasper's pincers. Force measuring tools were used to apply a known force at the handle, and the resulting force at the pincers of the grasper was recorded. The team used this data to develop a mathematical relationship between the two forces. In this way, the pressure applied by the grasper can be measured without interfering with the use of the FLS Trainer. The relationship was accounted for in the software, so that any changes can be implemented without replacing parts. As shown in *Figure 6*, Westebring used an applied force model to create equations for the relationship between grip force and force applied to the tissue [3]. This will be a helpful starting point to develop the applied force calculations.

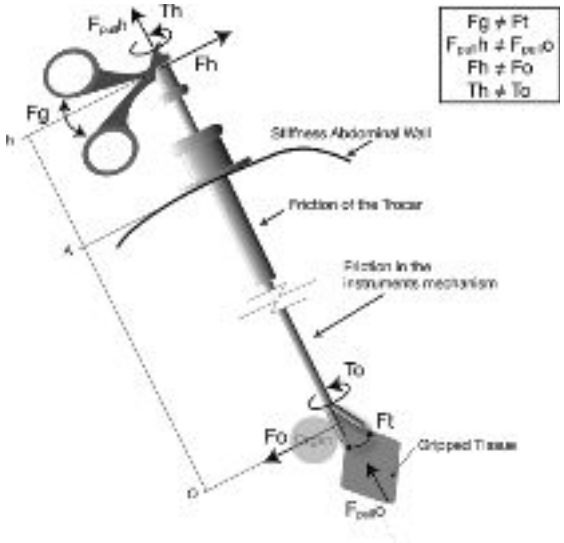

Figure 6: Grasper Tool Applied Force Calculation [3]

3.2.7 Training Box and set-up

Figure 7 and Figure 8 give a general layout for the FLS Trainer system, including the additions from this design.

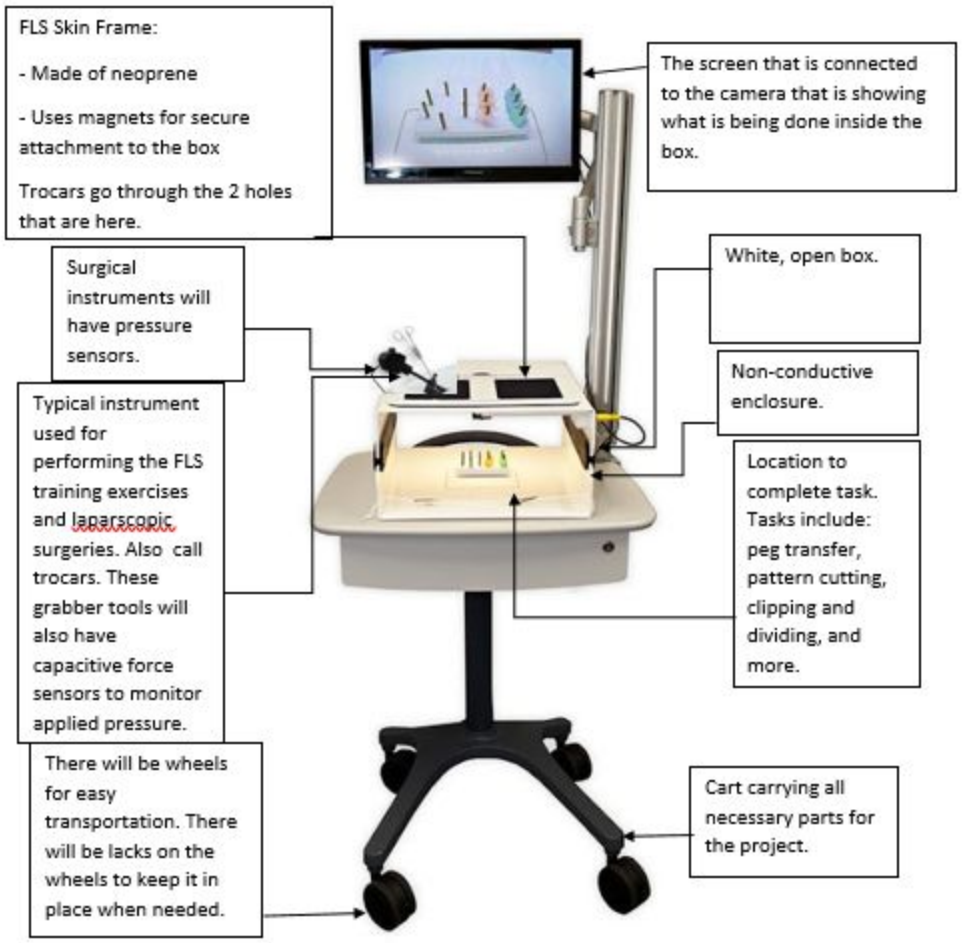

Figure 7: FLS System Layout

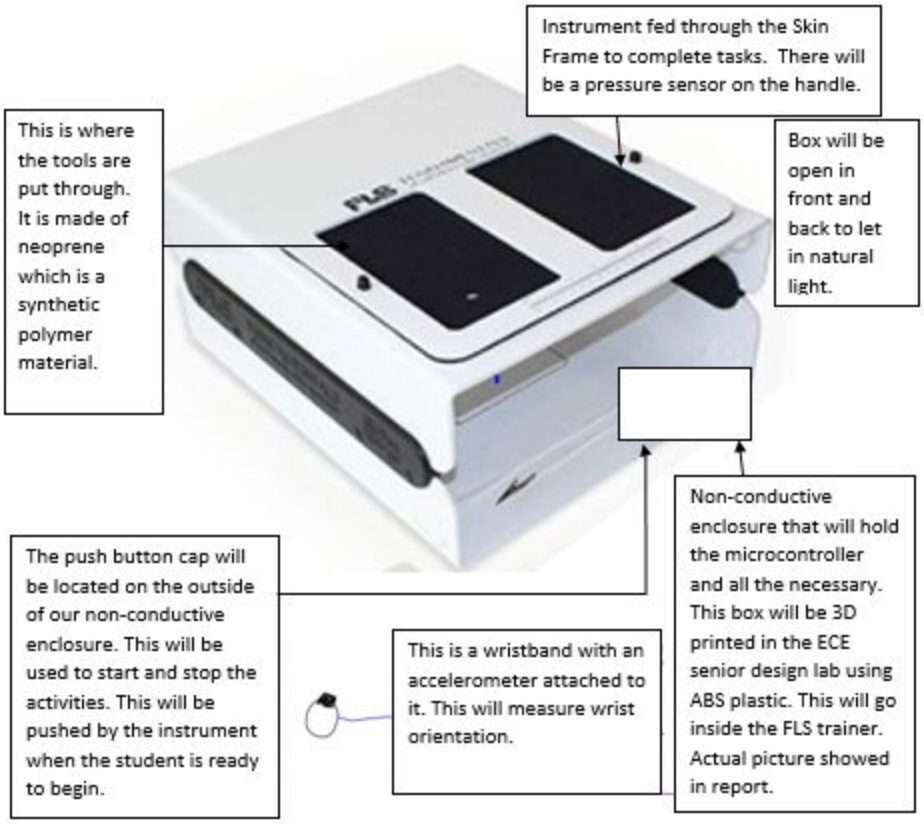

Figure 8: System Design Layout

4.0 Design and Performance

4.1 Schematics and Design

Figure 9 details the schematic for the entirety of the system. The connections from the Discovery board to the force sensor, accelerometer, and push button are all properly shown. Both force sensors are powered from the 5VDC bus and communicate along the same I^2C bus. Both accelerometers are powered from the 3VDC bus and send signals to three ADC GPIO pins each. The test push button and calibration buttons are wired to digital input GPIO pins. The LED indicator is wired to a digital output GPIO pin. The Discovery board and other components will all be put inside a non-conductive enclosure that will go inside the FLS training box. The push button will be on the outside of this enclosure able to be reached by the medical student's tools that are being used to complete the activity. The accelerometers will be on a wrist band on the student's wrists and the force sensors will be on the grasper tools, but both of them will be wired as shown.

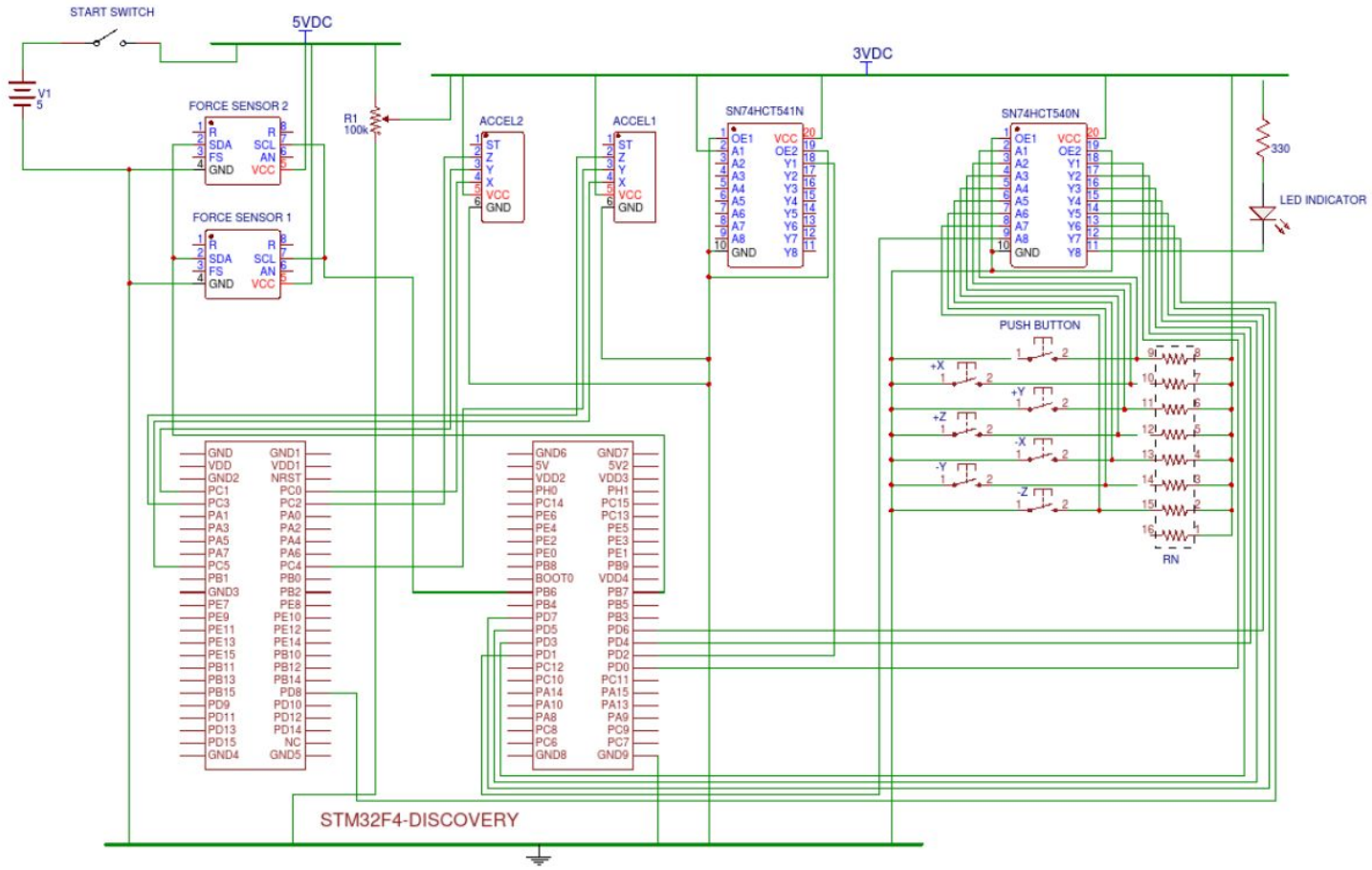

Figure 9: Schematic for the Entire System

4.2 System Integration

To properly complete this project the sensors and the timer must be recording data at the same time and outputting this data to a log while the student is doing the task. A program called STM Studio was used to output the data and display the variables in real-time. STM Studio can interface with the Discovery Board using ST-LINK development tools [8].

All of the components and circuitry, outside of the force sensor and the accelerometer, will be confined inside a non-conductive enclosure inside the FLS box trainer. The start button will be on top of this box so the students can push it with the grasper tool. The accelerometer will be on the student's wrists attached to a wrist band with cables going to the non-conductive enclosure inside the box trainer. The force sensors will be on the grasper tools with cables going to the non-conductive enclosure inside the box trainer. The particular pin assignments for each of the peripheral devices is detailed in *Table 3.* This setup will allow the students to know how

much force they are using, the orientation of their wrists, and how much time it took them to complete the task.

GPIOA

	7	6	5	4	3	2	1	0
Mode						AF2	AF1	
Module						TIM5_CH3	TIM2_CH2	
Function						TIME TRACK	ADC TRIGGER	

GPIOB

	7	6	5	4	3	2	1	0
Mode	AF4	AF4						
Module	I2C1_SDA	I2C1_SCL						
Function	FORCE 1-DATA	FORCE 1-CLK						

	15	14	13	12	11	10	9	8
Mode					AF4	AF4		
Module					I2C2_SDA	I2C2_SCL		
Function					FORCE 2-DATA	FORCE 2-CLK		

GPIOC

	7	6	5	4	3	2	1	0
Mode			AN	AN	AN	AN	AN	AN
Module			ADC1_CH15	ADC2_CH14	ADC1_CH13	ADC3_CH12	ADC2_CH11	ADC1_CH10
Function			ACCEL 2-X	ACCEL 2-Y	ACCEL 2-Z	ACCEL 1-Z	ACCEL 1-Y	ACCEL 1-X

GPIOD

	7	6	5	4	3	2	1	0
Mode	FL IN	FL IN	FL IN	FL IN	FL IN	PDOWN IN	FL PP OUT	FL IN
Module	-	-	-	-	-	-	-	-
Function	-Y CALIB	-X CALIB	ZX CALIB	+Y CALIB	+X CALIB	START SW IND	INDIC LED	PB INPUT

	15	14	13	12	11	10	9	8
Mode								FLOATING IN
Module								-
Function								-Z CALIB

Table 3: Discovery Board GPIO Bitmap

5.0 Software

The software portion of the sensor interface design can be divided into four separate parts. The first part consists of the pushbutton and the timer. The second part handles the accelerometer data. The third handles communication with the force sensors. The fourth handles communication with the PC. Below is a summary flow chart outlining the structure of the main program. Sections 11.1.1 and 11.1.2 contain the main and ISR .c files used to implement the system given the tools available. Sections 11.2.1 and 11.2.2 contain the main and ISR .c files used to implement the system given ideal conditions (two full sets of grasper tools, force sensors, and accelerometers).

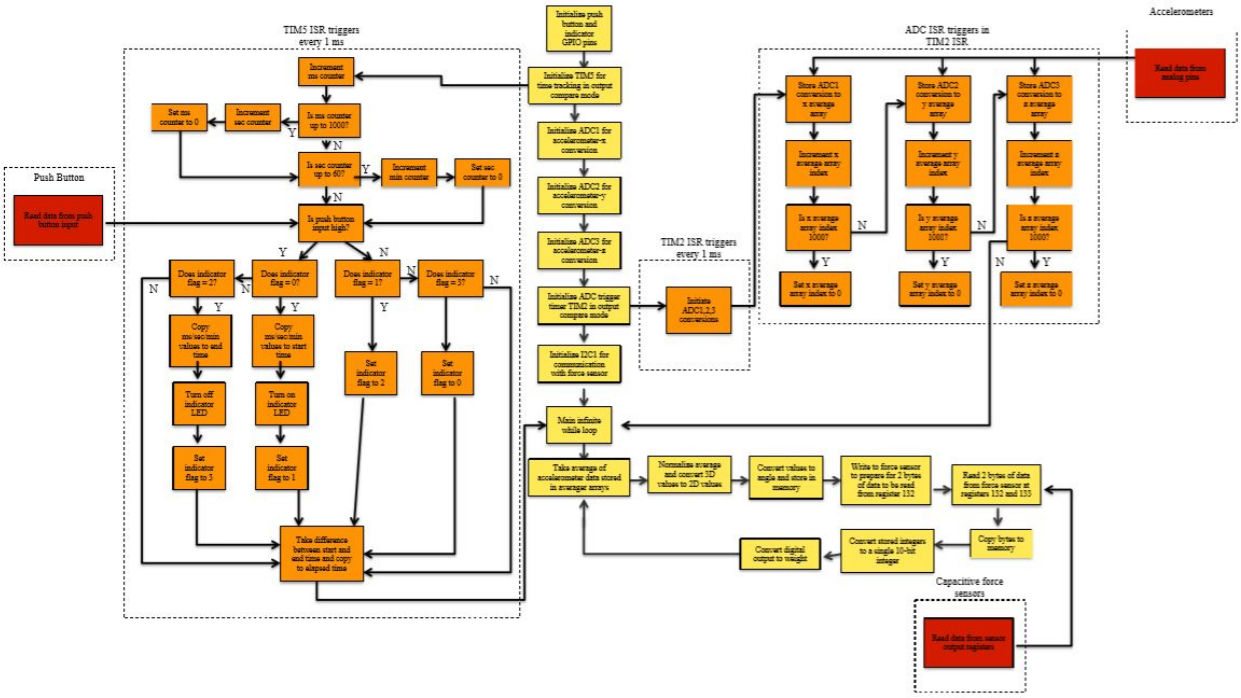

Figure 10: Summary Flow Chart of the Main Program

5.1 Timer and Pushbutton

The pushbutton is connected pin PD0, which is configured to digital input mode. This program uses the Discovery Board's timer module, specifically Timer 5. Timer 5 is a 32-bit timer that increments at a rate half that of the HCLK frequency. With the HCLK frequency set to 100 MHz, timer 5's frequency is 50 MHz. In output compare mode, Timer 5's interrupts can be triggered every 1 ms by configuring its period to coincide with the number of timer cycles that occur in 1 ms. The details of this calculation can be found in 11.1.1. When this interrupt is triggered, the Discovery Board jumps to the TIM5 ISR, which increments a millisecond counter, then uses this millisecond counter to create a standard counter that logs time in minutes, seconds, and milliseconds. In this same ISR, the routine checks the status of pin PD0 - the pushbutton

input. If the button has been pressed, the ISR saves the counter value to memory. This ISR continues to run every millisecond, and when the pushbutton is pressed a second time, the counter value is saved to a different memory location. The difference between these two times is taken and saved to a third memory location.

5.2 Accelerometers

This section of the software uses the Discovery Board's ADC modules to digitize the analog accelerometer signals. Since the accelerometers experiences a constant 1g acceleration downward, the x, y, and z components of the accelerometer outputs indicate how much of that 1g is experienced in each of the three directions. This information can be directly translated to an angle value. The three ADC modules are initialized to 12-bit resolution, conversion triggered by a timer 2 channel 2 interrupt, 3 cycle sampling, and 5 cycle delays between sampling. They are configured to handle two channels each, in independent scan mode. Each ADC handles the x, y, or z signals from both accelerometers.

The ADC Handler copies the conversions from each accelerometer to memory in an array of 1000 (1 per millisecond, so the array is completely refreshed every second) so that the main program can access these values. The main does most of the work of this program. It uses the array as the basis for an averager to minimize the effects of high frequency noise. The maximum and minimum values of each of the three (x, y, and z) signals are stored in the program, and these values are used to transform the average outputs into standardized values that can be compared to one another. Since the angle being measured is the angle of the x axis with respect to the horizontal about the y axis, the two values that are used to calculate the angle of the accelerometer are the x- magnitude and the magnitude of the vector sum of the y- and z- values. Once both of these magnitudes are calculated, the ISR uses the math.h library to find the arctangent of the ratio of these two values. This gives the angle of the accelerometer in radians, though some conditional logic is necessary to avoid error due to the limitations of the arctangent function because of the ambiguity of the sign of both arguments. This radian value is then transformed to degrees.

5.3 Force Sensors

The Discovery Board communicates with the force sensors using an I^2C bus. When using I^2C, there is usually one master device and up to 128 slave devices. In this project's case, the master device is the Discovery Board and the slave devices are the force sensors. The bus has two signal lines: SCL, which is the clock signal and SDA, which is the data signal. In general, the SDA line turns on or off when the SCL is low, and when SCL is high, whichever is the receiving device will read the data on SDA. The two exceptions to this are the start (SDA goes from low to high before SCL turns off) and stop (SDA goes from high to low before SCL turns off) sequences. These sequences indicate the beginning or end of a communication from a device.

Each of the slave devices on an I^2C will have a 7-bit (though sometimes 10-bit) address, hence the 128 (2^7) possible devices. This address is how the master device indicates that the slave is being communicated to. The default slave address for the SingleTact force sensor is 0x04, though it is reconfigurable. In general, any communication on an I^2C bus will be 8 bits of

data from one device to another, followed by an acknowledge bit from the receiving device to indicate that the instruction was understood. To write to a slave device, the following sequence of events must occur on the bus:

1. Send start sequence
2. Send I^2C slave address followed by a read/write bit 0 (when writing to a slave, this bit should be 0, when reading from a slave, this bit should be 1)
3. Send internal address of the slave's register to which the user will write
4. Send data to be written to this register
5. Send stop sequence

To read from a slave device, the following sequence of events must occur on the bus:

1. Send start sequence
2. Send I^2C slave address followed by a read/write bit 0 (when writing to a slave, this bit should be 0, when reading from a slave, this bit should be 1)
3. Send internal address of the slave's register from which the user will read
4. Send start sequence
5. Send I^2C slave address followed by a read/write bit 1
6. Read data from the specified register
7. Send stop sequence

The main c file in 11.2.1 was only implementable after reconfiguring the addresses of both sensors. As both sensors had a default address of 0x04, with both of them on the same I^2C bus, both would have responded to the same directives, and their outputs would have been indistinguishable from one another. After this readdressing side-steps this problem, the main file first initializes the Discovery Board's I2C1 module with the standard 100 kHz SCL speed, 50% duty cycle, and 7-bit slave address mode. Then, the communication standards specified above, along with information in the SingleTact user manual were used to create functions that could read or write to the force sensors as needed. The program uses these functions to read registers 132 and 133 of the force sensors' interface boards. These registers contain the most significant byte and least significant byte (respectively) of the board's 9-bit digital conversion of the force sensor's analog output. The valid range of this digital value spans from 0x100 to 0x2FF, which coincides with analog outputs from 0.5V to 1.5V, which mark the lower and upper barrier of the sensor's full scale range: 0 lbs and 22 lbs. The factory calibration ensures that these values indicate endpoints to a linear digital-value-to-force relationship. The program in 11.2.1 retrieves the digital output of the force sensor and linearly transforms this digital value to a weight measurement.

5.4 Calibrations

Every sensor requires calibration to be able display useful and accurate results. The variation that necessitates calibration may be a result of environmental differences from where the sensor was made to where it is being used, manufacturing variability, and noise effects. Every calibration requires a standard reference to be able to calibrate the sensor against. This

section details the calibration methods developed to ensure that the sensors and push button were accurately measuring [6].

The push button and timer system relies on internal clocks with constant, specified frequencies, so it did not require calibration in the traditional sense. Rather, the selected HCLK frequency and APB2 prescaler were used to determine what value should be programmed to the timer's auto-reload register. The rest of the process was internal to the Discovery Board. To verify that Timer 5 was accurately clocking milliseconds as they passed, and that the subsequent logic was accurately logging these increments as seconds and minutes as time passed, a stopwatch was used. The button and the stopwatch were initiated and stopped at the same time. The time elapsed as measured by the stopwatch was compared to that of the Discovery Board to ensure that the clock, prescaler, auto-reload, and interrupt settings all resulted in an accurate measurement of time in minutes, seconds, and milliseconds.

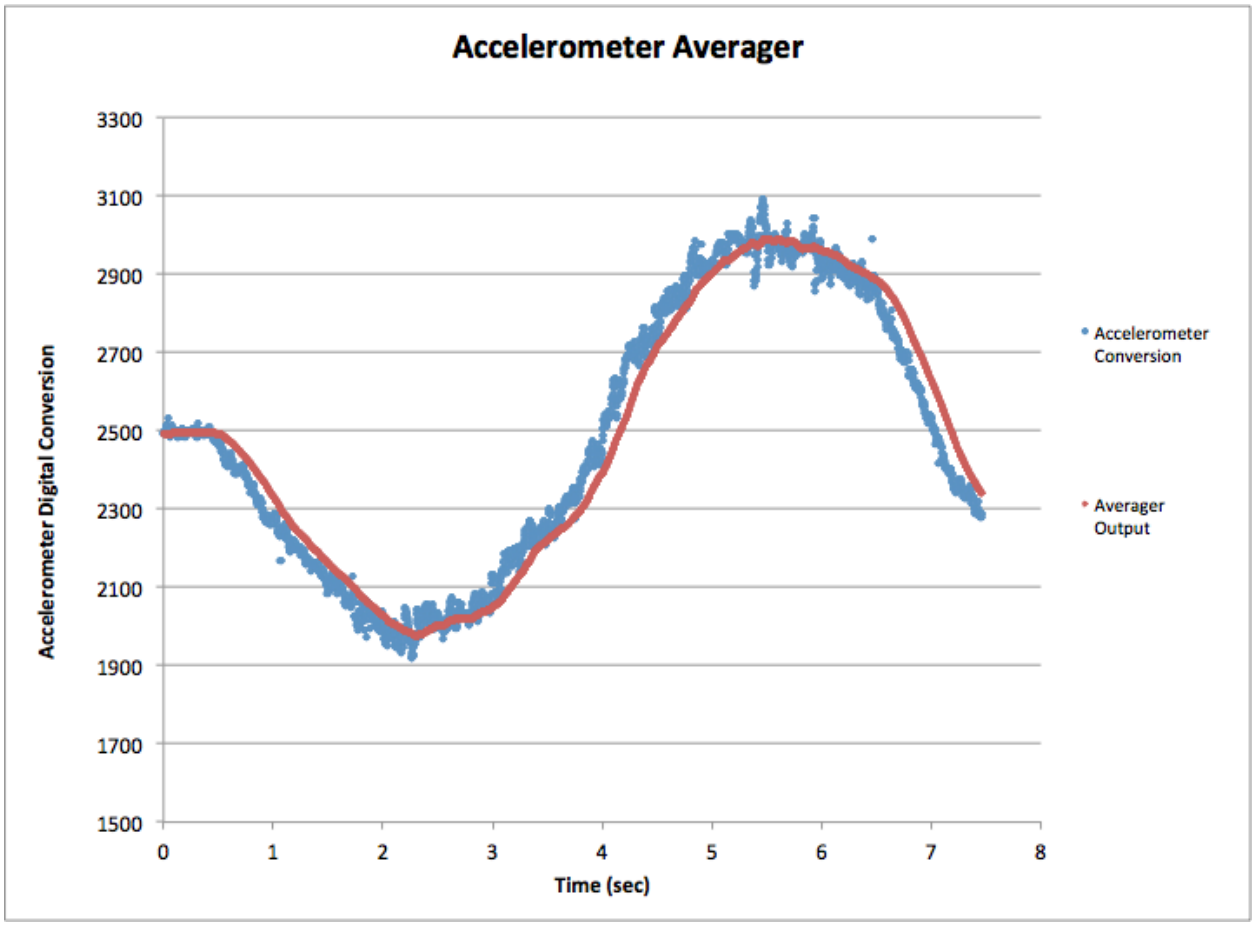

Figure 11: Accelerometer Averager

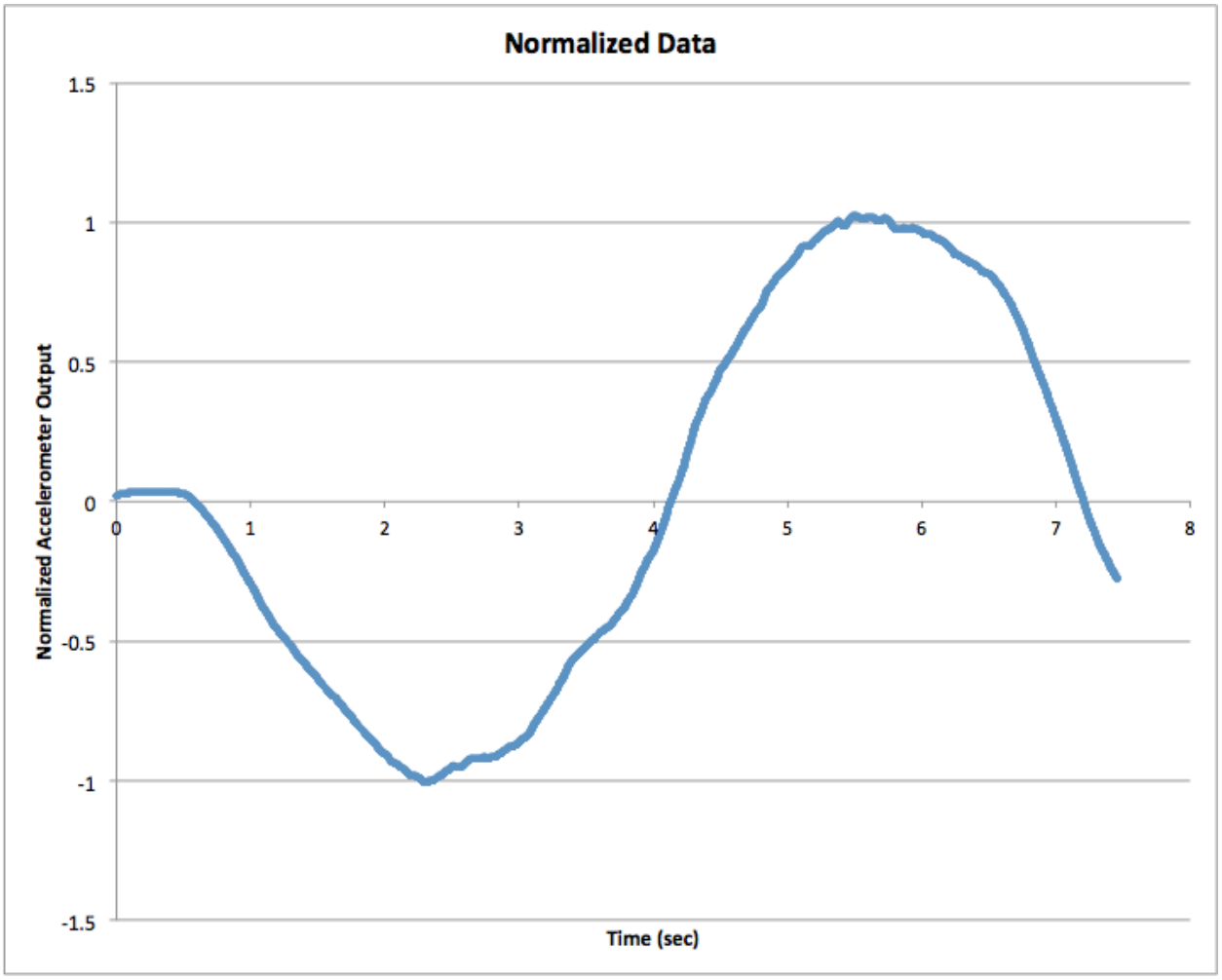

Figure 12: Normalized Data

The calibration of the accelerometers required a bit more intervention. This sensor is internally calibrated during production, although mechanical misalignment can negatively affect the accuracy of the sensor's calibration. The mechanical misalignment can be calibrated out of the system. These accelerometers do produce analog outputs proportional to the acceleration they experience, but the X, Y, and Z outputs are not proportional to one another. That is to say, if the X and Y axes both experience 0.6g, the voltage at their output pins will fit their respective voltage vs. acceleration curves, but their values will not be equal. To properly calibrate these sensors, the signals were conditioned to be proportional to one another. Assuming no external disturbances, the voltage at any of the three outputs is at a maximum when its respective axis is experiencing 1g (the unit vector pointing straight up from the ground). The outputs are at a minimum when the axes are experiencing -1g (the unit vector pointing straight to the ground). To begin the calibration process, the digital conversions of the sensor outputs in each of these positions are saved to memory (maxX, maxY, maxZ, minX, minY, minZ). Subtracting the axes minimums from their maximums yields the range of outputs that the sensor will output in

response to angle changes. These ranges are used to normalize incoming digital conversions to values between -100 and +100. This normalization accounts for any disproportion in the sensors' analog outputs, and because the normalized values are centered about 0, they can be immediately applied to the trigonometry that yields the angle measurement. To confirm this angle measurement and check to see if the sensor was properly calibrated, a protractor was used to situate the sensor at different angles. The first test began at 0° and subsequent tests checked angle measurements every 45° to makes sure the sensor was supplying accurate data [7]. In addition to checking the accuracy of the final angle calculation, the analog output was also checked at the sensor terminals and their corresponding digital conversion values corresponded linearly to the acceleration they experienced. (There was no way to directly measure the acceleration experienced by the sensor. Instead, a level and protractor were used to determine the exact angular position of the accelerometer with respect to the direction of gravity. Then the x, y, and z vector components of the 1g vector the accelerometer experienced directly downward were determined using trigonometry.)

Like the accelerometers, the force sensors are calibrated during production, and the I^2C interface boards are pre-calibrated and matched with their sensor to minimize any deviation from the ideal linear output. This provides better accuracy and linearity than a standard sensor. A digital scale was used to confirm that the sensor was properly calibrated. The sensor's output varies from 0 V to 2 V DC in response to applied force. The interface board converts this analog output to a 10-bit digital output from 0x000 to 0x3FF. Analog output voltages from 0.5 V to 1.5 V correspond to the sensor's full scale range (FSR). The sensor used in this design is rated for up to 22 lbs., so the full scale range varies from 0 lbs to 22 lbs. The corresponding FSR output of the interface board varies from 0x100 to 0x2FF. These weight-output relationships were used to convert the interface board's output from a 10-bit integer to a weight. To verify that this method was accurate,weight was applied to the force sensor, using the digital scale to independently measure the applied weight. The analog and digital outputs of the sensor and interface board were scaled according to the FSR and compared to the scale reading [5].

6.0 Bill of Materials

Table 4: Bill of Materials

Item	Number ordered	Part number	Cost
Accelerometer	3	ADXL335	$44.85
Force Sensor	2	S8-100N	$149.90
Discovery Board Microcontroller	1	STM32F407VG	$19.90
8 Pin Male DIN Connectors	3	CP-1080-ND	$4.20
8 Pin Female DIN Connector	3	CP-3180-ND	$4.11
O Ring Kit	1		$8.00
Push Buttons	100	PTS645	$10.41
		Total	$241.37

7.0 Conclusions

Table 5: Specification Results

Specification	R, G, or P	Result	% complete
Sensor able to accurately measure pressure from 0.1 to 4 PSI	R	The force sensor is currently being worked on to measure to the nearest tenth.	90
Sensor able to accurately measure a radial orientation range of 210º	R	The accelerometer is capable of measuring wrist orientation for 0° to 180° and 0° to -180°, measuring a total of 360°. This is almost done being calibrated	100
Sensor able to mark the beginning and ending of an FLS Trainer test	R	The push button mark the start and stop times of the test. The difference between the two is calculated to display the total elapsed time.	100
All sensors must weigh less than 1 oz	R	All sensors used weigh less than 1 oz.	100
Discernable analog or digital signals from sensors to microcontroller	R	All signals are being transferred from the sensors to the microcontroller accurately.	100
Microcontroller system STM32F4 Discovery Board	R P	The microcontroller is used to integrate all sensor data into one program. The STM32F4 was the microcontroller used for this project.	100 100
5 VDC power source	R	A 120 VAC to 5 VDC power converter is used to power the circuit and microcontroller.	100
C programming	R	The language used to program the STM32F4 is the C programming language.	100
Serial connection required from microcontroller to PC	R	The ST-Link is being used.	100
Data stored in accessible format Excel file	R P	Using STMStudio to export it to an excel file	100
The microcontroller and other auxiliary electronics should be contained in a nonconductive	R	The microcontroller and the other electronics fit in the nonconductive enclosure that was 3D printed.	100

enclosure			
The enclosure should be mounted on the FLS Trainer cart	R	The nonconductive enclosure is will be velcroed to the inside of the training box.	100
The enclosure should weigh less than 1 lb	G	The nonconductive enclosure the was 3D printed is less than a pound.	100
Components should ideally require 3.3 VDC or 5 VDC power	G	All components that require voltage not in this range will be stepped up or stepped down.	100

Table 4 above displays the results of each specification that was given in this project.

7.1 Performance Measurement and Test Results

The actually testing procedure was designed to be straightforward and easy for the user to understand. The designed non-conductive enclosure fits easily in the box trainer behind the task that is setup. Wires for the sensors, power, and computer communications run from the enclosure, outside the back of the box trainer, and then up to the grasper tools. The following is the standard order of procedure the user must abide by to ensure proper operation:

1. Put on the wristband so that the accelerometer is comfortably positioned on the wrist, below the thumb.
2. Flip the main power switch to the “on” position to supply power to the external circuit.
3. Hold the wrist in the base position (with thumb facing up).
4. Press buttons labeled “X” to calibrate the accelerometer x-axis base positions.
5. Press buttons labeled “Y” to calibrate the accelerometer y-axis base positions.
6. Press buttons labeled “Z” to calibrate the accelerometer z-axis base positions.
7. Grab the grasper tools and hold them in ready position.
8. Press push button cap with right grasper tool to begin data collection when ready to start task (green LED will turn on to show that test in session).
9. Perform the tasks necessary of the box trainer procedure.
10. Press push button cap with right grasper tool to end data collection when ready to finish task (green LED will turn off to show that test is no longer in session).

During and after the testing procedure, the user can attain sensory feedback using STMStudio. While the box trainer task is being performed, STMStudio can be set up so that the user can view sensory data real time. After the task is completed, the collected data can be viewed using STMStudio, or exported as a log file so it can be analyzed using another program.

Each specification has been met with minimal deviations to how it was originally detailed. There was a change in how the microcontroller is communicating with the PC. Instead

of using a serial connection STLink is being used, which is an in-circuit debugger and programmer for the Discovery board. Also, instead of exporting the data to an excel document, STMStudio is being used. All of the given specifications have been attempted and completed. The most underdeveloped of the specification results is the approximation of the force applied by the grasper tool as the force applied at its handle. This approximation could be improved by determining the relationship between the force applied at the handle where the force sensor is and the force applied at the end of the tool using two force sensors. The relationship is rather linear, but outside factors such as the topography of the object being manipulated by the grasper tool or how the medical student holds the grasper tool can effect the force applied at the end of the tool.

The final, non-conductive enclosure is shown in Figure 13. This holds the Discovery board, the push button that will start the timer, and all the other external circuitry. One the top of the enclosure is an LED that will show while the timer is running and the push button cap that can be easily reached by the grasper tool to start the task. On the side of the enclosure is the start switch that will energize the entire system along with the six calibration buttons used to calibrate the accelerometer.

Figure 13: The Non-conductive Enclosure

Overall, the microcontroller was programmed successfully to gather and coordinate the sensory interfaces, implemented the capacitive force sensor and interface board in accurately measuring the force on the grasper tool, converted the accelerometer data to wrist orientation, and used STMStudio to read and record sensor interface information.

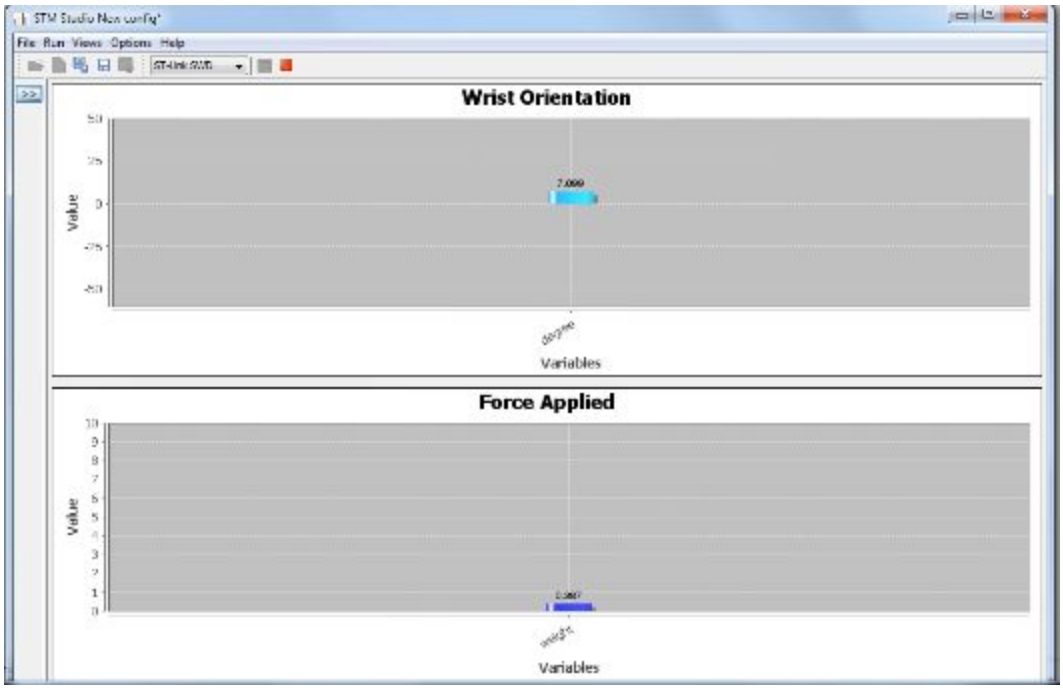

Figure 14: STMStudio Output Display

7.2 Concluding Statements

At the completion of the design project, several items will be turned over to the sponsor as elements of the whole design. Hardware items delivered will include an augmented FLS Trainer system, a nonconductive enclosure containing vital equipment, and hardwire connections between these elements. The augmented FLS Trainer system consists of a standard box trainer with the addition of sensors. There will be motion sensors permanently mounted to the inside of the FLS Trainer box, force sensors permanently connected to the inside trainer's grasper handles (positioned so as to not obstruct students' hand movements during use), and wristbands with calibrated inclinometers to be worn while using FLS Trainer. The nonconductive enclosure will include an STM32F4 Discovery Board. The sensors will be connected to the ADC pins of the Discovery Board using 28 AWG signal cabling. The Discovery Board will be connected to the FLS Trainer PC using a male to male USB cable.

The software component of the design will be directly written to the Discovery Board, but a copy of the original code (in C) will be made available to the sponsor for reference and possible future editing. The code will be thoroughly annotated to maximize readability and facilitate any necessary changes should the need arise.

Supplementary deliverables include complete literature and ordering information for all three sensors, the STM32F4 Discovery Board, and all signal cabling. Contact information for all team members, the Project Proposal, and the final project paper will also be available to the sponsor, should he request it.

7.3 Recommendations for Further Study

There are several improvements that could be implemented into this project that could enhance the performance and usefulness of this project. One suggested upgrade would be to use both an accelerometer and gyroscope to accurately record wrist orientation. By just using an accelerometer, the data will not be completely accurate. The accelerometer is sensitive to vibration and noise, especially high frequency noise. Since a gyroscope measures rotation, it is less sensitive to the linear mechanical motions that affects the accelerometer. However, gyroscopes are affected by different problems such as low frequency noise and drift. Once the gyroscope and accelerometer data is averaged, the results become more accurate [9].

The design could also be improved by having the accelerometer and force sensor wirelessly communicate with the microcontroller. This would allow a greater range of motion for the test taker and it would make the test feel more like actual surgery. This would require, however, small transmitters and receivers to be utilized along with the microcontroller and the sensors. Making improvements to the test, while causing little to no change in the procedure, is the ultimate goal for a project like this.

Another improvement that can be made is to increase the precision for force sensor by using the digital scaling value. It is possible to make the digital scaling value more accurate by adjusting it in 0.01 increments. This would be useful so the practicing student can have the most accurate data available.

Currently, the push button timer only measures the start and stop time to tell the practicing student how long they took to complete the task. A continuously updating timer would be the next logical step. The student should be able to have the option to know at all times how long it has taken them to complete the task thus far.

Finally, a self-test and calibration board could be very useful to receive the most accurate results and to allow the student to easily set up the activity. The user would interface with the program using the LEDs and switches. These LEDs and switches would be arranged on a board, as seen in Figure 15 for clarity. Once the sensors have been initially calibrated, the program would run a check on the signals received from the sensors immediately after the power switch are both turned on. This check would compare the signals with the calibrated values stored as arrays in the program. If the self-check yields any concerning results, the corresponding ‘Calibration Mode’ LED will either flash continuously (indicating a possible loose connection) or blink (indicating that the sensor likely needs recalibration). At this point, based on the results of the self-test, the user can use the switches to indicate whether she/he will calibrate one of the sensors or proceed with the FLS trainer test (the Data Acquisition Mode in the software). *Table 5* details the switch sequences recognizable by the program. The sequence of the switches will indicate which function to proceed with.

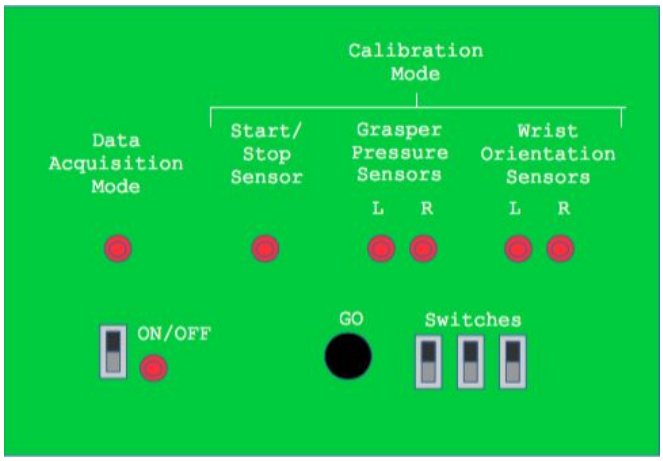

Figure 15: Self-Test and Calibration Board

Table 6: Switching Calibration

Switch Sequence	Procedure
000	Data Acquisition
001	Calibrate Touch Sensor
010	Calibrate Left Pressure Sensor
011	Calibrate Right Pressure Sensor
100	Calibrate Left Inclinometer
101	Calibrate Right Inclinometer

8.0 Literature Review

Several different search engines for research were utilized including WMU library databases and Google. All gathered resources were selected to provide detailed information about specific elements of the design. This section contains summaries of each item of literature and a brief overview of how they contribute to the progress of the design project.

"What is Laparoscopic Surgery" was obtained to gain general insight on what laparoscopic surgery is and how it is beneficial. For this design project, the purpose is to better equip surgeons for surgery, so a thorough understanding of laparoscopic surgery is a necessity [1].

Laparoscopic surgery has its limitations, however, and in some situations, inhibits the surgeon from being able to get their hands on specific organs. Since it is difficult to copy what the human hand is capable of in surgery, a lot of surgeries cannot be performed using laparoscopic methods. If surgeons were able to have more control over, as well as a better understanding of, the applied force by the instruments used during the procedure, it may improve

the range of surgeries that could be completed [1]. This information supplies the designers with a higher understanding of what specifications the design will need to satisfy.

To further specify the importance of what needs to be designed for this project, areas of imminent improvement were examined. In “Haptics in minimally invasive surgery - a review”, feedback was researched to try to determine the largest disadvantages of laparoscopic surgery. Of the several issues, the biggest was the inability to gauge the applied force of the grabber tool on tissues. The way to determine the applied force at the end of tool based upon the force at the grip was calculated [3]. This information will be helpful in selecting the correct capacitive force sensors to use in the design.

Although the importance of monitoring the applied force on the tissues was stressed throughout the article, a solution was not considered. This opens the door to be creative and consider if this design can satisfy the requirements of the desired in this article.

The design is centered around making improvements to the Fundamentals of Laparoscopic Surgery (FLS) intelligent surgical box trainer. In “FLS Trainer System and Accessories,” the FLS Trainer System and its importance was briefly explained. The trainer System is essentially a device “that facilitates the development of psychomotor skills and dexterity required during the performance of basic laparoscopic surgery [2].”

It was important for this information to be gathered so that readers can have a solid explanation of what the design is centered around. This helps develop a background knowledge of the FLS system and laparoscopic surgery. Having a solid background knowledge of information discussed in this report makes the concepts and designs easier to understand.

To further understand the development of the intelligent surgical box trainer, it is helpful to understand the history of development of the FLS educational trainer system. Since new improvements are to be made on the box trainer, a thorough background understanding of the system is important. The FLS educational system was developed in 2004 to help educate surgical residents and practicing surgeons in a safe and effective manner. Since that time, over 9,000 people have been certified to perform laparoscopic surgery through the program. Over the years, the surgical box trainer has been integrated with growing technology to further improve its ability to educate. With many designers creating new ways to improve the system, the effectiveness of the program has continued to increase [4].

Simulations are used in many different fields to educate students and trainees in a situation similar to what they will be going through in their field. In [10], the authors specifically talk about simulations needed in surgical education. This is needed to give patients better care. Surgical students should not have to learn on real people if they can perfect their skills through simulations. This document also talks about other positives of simulations such as students can practice on their own time and it can cost less than getting time to practice in an actually operating room [10].

The paper then discusses how the laparoscopic surgery simulations still need improvements. The user gets little feedback and the image seen on the observation screen is two-dimensional. It also mentions that there are both high-fidelity and low-fidelity simulations for laparoscopic surgery. The FLS Trainer box is considered a low-fidelity training simulation, but it is still very helpful in training surgical students. The training box is portable and can be used at home for practice. Some simulation exercises that can performed with this training box include peg transfer, pattern cutting, clipping and dividing, and more. The more precise and fast

the student is, the higher the student's score will be. Multiple studies have shown that practicing with the box trainer and other simulations improve skills [10].

The document used to research the preferred microcontroller for this design is the User's Manual to the STM32F4 Discovery board. It provides a general explanation for what the Discovery Board is, its functions, how to use it, and what it includes. The Discovery Board has a high performance microcontroller. The user can easily develop programs in C and implement them with this board. It has an ARM Cortex M4 32-bit core with a 1 Mbyte Flash Memory and a 192-Kbyte RAM (random access memory) [11].

One item that the board includes is a ST-LINK/V2 embedded debugging tool. It also has a 2 ST MEMS (microelectromechanical sensors) digital 3-axis accelerometers. MEMS can include inclinometers, digital compasses, inertial modules, force sensors, humidity sensor, and microphones. Other things the board includes is a digital microphone (audio sensor), one audio DAC (Digital-to-Audio Converter), 8 LEDs, two push buttons, Ethernet connectivity, and USB (universal serial bus) OTG (on the go) FS (full speed) with micro-AB connectivity. Of the 8 LEDs, one is used for USB communication, one is used for power on, four of them are available for the user to program, and two of them are used for USB OTG. Of the 2 push buttons, one of them is labelled USER and the other is labelled RESET. The board's power is supplied through the USB or external power sources of either 3V or 5V. The Board comes with STM32 comprehensive software HAL (Heuristically programmed ALgorithmic computer) but can also interface with other software packages [11].

To use the Discovery board the user must have a Windows operating system to program (either XP 7, 8, or 10) and a USB type A to a Mini-B cable [11].

The User's Manual for the Discovery Board describes many things such as how to get started, the system requirements, and the development toolchains that are supported. Toolchains are software development tools. The user manual also describes the hardware and the layout of the board. The board has a 100-pin LQFP (Low-profile quad flat package), which is an integrated circuit that has high I/Os while also having low profile requirements. I/Os either transfers data to or from a peripheral choice or computer. There are many figures to describe the layout of the board. It also talks in depth about the embedded ST-LINK/V2, the drivers, the power supply and power selection, and the pin outs. It explains the audio capability, the OTG, the motion sensor, the jumper (JP1), and the OSC (oscillator) clock supply [11].

Grand Valley State University students conducted a study on the Fundamentals of Laparoscopic Surgery (FLS) Trainer box and force sensors. They found a need to make improvements on the hardware and software of the box. With the FLS Trainer box, there is no way to record and display the forces applied by the surgeon during the task. Tracking force is important because if less force is used during surgeries outcomes generally show better results. When less force is used, there is less tissue damage and less blood loss [12].

In this study, the students added force sensing capabilities and a hand sensor module. The LCD display and original modules were not changed. The main board platform that was in the box was changed so that there was a force sensor on each of the four corners. With these sensors, relative force magnitudes could be measured at each corner. The students also added a wireless transceiver to the main board. With this it can send signals between itself and a new hand sensor that was also added by the students [12].

The hand sensor was a main change that the Grand Valley State students added to the FLS Trainer box. They wanted to add a hand sensor to measure the handedness. Handedness is

the use of one hand over the other. They added the wireless hand sensor module to one of the surgical instruments. This is so they could tell which hand was being used and when the hand was used. This module has a wireless transceiver, a chip antenna, an accelerometer, and a microcontroller communicating between the wireless chip and the accelerometer. The instrument that it is attached to is like the ones surgeons use. The four sensors that are on the four corners of the main board can interact with the hand sensor to tell which hand is being used and what force is being applied [12].

Arizona State University students conducted a study on the Fundamentals of Laparoscopic Surgery (FLS) Trainer box and motion sensors. They wanted to add a multi-sensor feedback system that would give the surgical students feedback while they were going through the training. This would improve the training process [13].

The main part of this paper talked about the hardware changes that they made. They added two more cameras to look at the hand movements. One camera was a USB webcam and the other was a kinetic camera. There were also data gloves that could be used to contribute to motion tracking. There was also a PC used to collect all the data and a frame grabber used to connect the video camera to the PC [13].

With these additions, the surgical students can look back at their work and know how they performed. To do so they must register and log in. By logging in, the training system can then know whose results are whose. The motion data collected can be graphed on the computer. Data can be collected from each test a surgical student does. Past graphs can be compared to their present test to display if the surgical students are getting better. There is also live-feedback collected, such as skill-related attributes, velocity, and smoothness [13].

One of the fundamental requirements of the design is to be able to indicate when the student using the FLS Trainer is applying too much pressure on the grasper. Though it is difficult to define the exact pressure at which human tissue will be damaged if the pressure is applied (the number will vary based on tissue type and strength, length of time exposed to the pressure, material applying the pressure, etc.) in general, a laproscopic surgeon should not exceed a force of 2.2 pounds per square inch on his or her grasper [14]. The pressure applied by an FLS Trainer user can be determined using a simple force sensor. However, there are several technologies employed in pressure sensing, each with their benefits and drawbacks. The most commonly used technologies are potentiometric, capacitive, inductive, and piezoelectric force sensors [15]. The sponsor for this design, Dr. Grantner, has indicated that capacitive force sensors will be the most accurate, robust, and cost-effective option for this design.

In order to develop a method for tracking the wrist orientation of a student using the FLS Trainer, two items needed further inquiry: how are wrists expected to move during this exercise, and what is the appropriate method to measure such movements? In the course of completing an FLS Trainer test, it is reasonable to assume that the motion of one's hands will be the result of elbow and wrist movements only. An index detailing the normal ranges of joint motion indicates that the human elbow can move the hand (about the axis of the forearm) a total range of 180°, and that the wrist can move the hand a further 20° [16]. This would indicate that a student's wrist orientation can move about a 200° range on average. A 3-axis accelerometer is a device that can measure the static acceleration of gravity in every direction as well as the dynamic acceleration that can be from motions or vibration. This is the instrument this design must utilize to catalog the wrist orientations of students completing the FLS Training System. The accelerometer being

used in this project will be small and low-powered to fit the requirements. It measures acceleration with a minimum full-scale range of ±3g. [7].

9.0 Alternative Designs

Coming up with alternative methods to complete a design is an essential part to every project. The paragraphs below discuss alternative microcontroller boards and sensors that could be used to complete this project if the original products chosen were to fail.

One alternative for the STM32F407VG Discovery Board microcontroller is the STM32F411VET6 Disco Board microcontroller. These microcontrollers are from the same manufacturer and are very similar. One of the main differences is that the Disco board has an additional 3D digital linear acceleration sensor and a 3D digital magnetic sensor. These additional sensors are not necessary for this project. If there was a need for these sensors in a later design, the Disco board microcontroller could be used.

Another alternative for the Discovery Board could be Amtel's Inertial One Sensor Board AVRSBIN1. This microcontroller has all of the sensor interfaces that would be needed to complete this project. The datasheet describes one of its main features as having a full nine-degree-of freedom inertial sensing. The software and development tools are readily available through Atmel. However, the Discovery Board was chosen for reasons such as the 8 LEDs that will assist in programming and knowing if the board is responding, the three different interfaces available, and the features that make the board easy to debug and program.

There are also possible alternatives to the accelerometer sensor used. Instead of using an accelerometer, an inclinometer could have been used. An inclinometer is an instrument capable of measuring angles between body parts that may be used to examine the motion of the body [17]. A possible inclinometer that could be selected based on provided specifications is a ADIS16203/PCBZ-ND Programmable 360° Inclinometer with evaluation board. This instrument could be utilized in this design to catalog the wrist orientations of students completing the FLS Training System. An inclinometer develops a signal using an internal pendulum, which varies based on the inclination of the device [18]. An inclinometer was not used in this design because the accelerometer could be used in the same way and the accelerometer was less expensive.

There is also room for expansion in eliminating the use of STMStudio for information storage. The Discovery Board has four universal asynchronous receiver/transmitter (UART) modules available for serial communication if the USB is dedicated to another task. The Discovery Board and the local FLS Trainer PC could communicate using a USB to serial UART cable. The UART modules and conversion cable could be used to send the sensor and timer information to the local PC independent of any outside software.

10.0 References

[1] "What is Laparoscopic Surgery." Internet: http://www.surgery.usc.edu/divisions/tumor/pancreasdiseases/web%20pages/laparoscopic%20surgery/WHAT%20IS%20LAP%20SURGERY.html [Oct. 30, 2016].

[2] “FLS Trainer System and Accessories.” Internet: http://www.flsprogram.org/testing-information/trainer-box/ [Oct. 30, 2016].
[3] E.P. Westebring-van der Putten *et al.* (2008, Feb.). “Haptics in minimally invasive surgery - a review.” *Minimally Invasive Surgery.* [On-line]. 17(1), pp. 3-16. Available: http://www.tandfonline.com/doi/full/10.1080/13645700701820242 [Oct. 29, 2016].
[4] L. Michael Brunt. “FLS: Celebrating a decade of innovation in surgical education.” *American College of Surgeons,* vol. 99, pp. 10-15, Nov. 2014.
[5] *SingleTact User Manual*, Pressure Profile Systems Inc., Los Angeles, CA, 2016, pp. 04-06.
[6] B. Earl, “Calibrating Sensors,” *Why Calibrate? | Calibrating Sensors | Adafruit Learning System*, 18-May-2015. [Online]. Available: https://learn.adafruit.com/calibrating-sensors/why-calibrate. [Mar. 28, 2017].
[7] Analog Devices, “ADXL335 Datasheet.” Internet: https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf. 2009. [Mar. 28, 2017].
[8] “STM-STUDIO-STM32 ,” *STM-STUDIO-STM32 - STM Studio run-time variables monitoring and visualization tool for STM32 microcontrollers - STMicroelectronics - STMicroelectronics*, Mar-2016. [Online]. Available: http://www.st.com/en/development-tools/stm-studio-stm32.html. [Mar. 28, 2017].
[9] “A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications.,” *Starlino Electronics*, 29-Dec-2009. [Online]. Available: http://www.starlino.com/imu_guide.html. [Mar. 28, 2017].
[10] S. L. de Montbrun and H. MacRae. (2012, Sept.). "Simulation in surgical education," *Clinics in Colon and Rectal Surgery*. [On-line]. 25(3), pp. 156–165, Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3577578/ [Oct. 30, 2016].
[11] STMicroelectronics , "UM1472 User manual Discovery kit with STM32F407VG MCU." Internet:http://www.st.com/content/ccc/resource/technical/document/user_manual/70/fe/4a/3f/e7/e1/4f/7d/DM00039084.pdf/files/DM00039084.pdf/jcr:content/translations/en.DM00039084.pdf Feb. 2016 [Oct. 30, 2016].
[12] P. Shields, "A Novel Electronic Laparoscopic Training Device." M.A. theses, Grand Valley State University, Allendale, Michigan, 2012. [On-line]. Available: http://scholarworks.gvsu.edu/cgi/viewcontent.cgi?article=1019&context=theses [Oct. 30, 2016].
[13] Q. Tian, L. Chen, Q. Zhang, and B. Li, "Enhancing Fundamentals of Laparoscopic Surgery Trainer Box via Designing A Multi-Sensor Feedback System," presented at Medicine Meets Virtual Reality Conference, San Diego, California, 2013. [On-line]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EC0CD98745DB41D26284A05D55648C4D?doi=10.1.1.727.6692&rep=rep1&type=pdf [Oct. 30, 2016].
[14] J.L. Grantner, Private communication, October 2016.
[15] P. Welander, “Pressure Sensor Technologies,” *PACE (Processing & Control Engineer)*, Vol 00(00), pp. 20, May, 2008.
[16] B.D. Appleton. (1996, January 09). *Normal Ranges of Joint Motion* (V 1.36) [Online]. Available: http://web.mit.edu/tkd/stretch/stretching_8.html#SEC84.
[17] *Taber’s Cyclopedic Medical Dictionary.* F.A. Davis Company, 2013, “Inclinometer”.
[18] D. Welch, “Systems Integration for Biosensing: Design, Fabrication, and Packaging of Microelectronics, Sensors, and Microfluidics,” *ProQuest Dissertations Publishing*, 2012.

11.0 Main Code

11.1 Final code

The following contains the program used to implement the design.

11.1.1 Final main .c code

```
/* Box Trainer Main */

/* Header Files */
#include "main.h"
#include "math.h"

/* Definitions */
GPIO_InitTypeDef GPIO_InitStructure;
NVIC_InitTypeDef NVIC_InitStructure;
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;
ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
I2C_InitTypeDef I2C_InitStructure;


void GPIODConfig(void);

void TIM2Config(void);
void TIM5Config(void);
void ADC1Config(void);
void ADC2Config(void);
void ADC3Config(void);
void I2C1Config(void);
void AccelCalib(void);
float AngleCalc(int levelX[1000], int levelY[1000], int levelZ[1000]);
void Write_SingleTact(void);
void ReadRQ_SingleTact(void);
void Read_SingleTact(void);
void command_out(int c);

extern int levelX1[1000], levelY1[1000], levelZ1[1000], levelX2[1000], levelY2[1000],
levelZ2[1000];
int Cx[4], sumX=0, sumY=0, sumZ=0, avgX, avgY, avgZ;
int minX=3460, minY=1402, minZ=1185;
int maxX=3804, maxY=2777, maxZ=3154;
int rangeX, rangeY, rangeZ;
float zeroX, zeroY, zeroZ;
float valX, valY, valZ;
float x2, y2, z2, result, ratio, angle, degree, degree1, degree2 store;
int WST_reg, WST_n, WST_data, RRQST_reg, RRQST_n;
float weight1, weight2;
/* SingleTact slave address = 0x05 for FS1, 0x06 for FS2
   when writing to TCN75A, R/W bit is 0 -> address 0x08
   when reading from TCN75A, R/W bit is 1-> address = 0x09
   the R/W bit is sent automatically using stm32f4xx_i2c.c */
uint8_t SingleTact = 0x0A, data_MSB, data_LSB;
uint16_t output1, output2;

int main(void)
{

  GPIODConfig();
  TIM2Config();
  TIM5Config();
```

```
  ADC1Config();
  ADC2Config();
  ADC3Config();
  I2C1Config();

  while(1)
  {
    /* ACCELEROMETER OPERATION */

    AngleCalc(levelX1, levelY1, levelZ1);
    degree1 = degree;

    AngleCalc(levelX2, levelY2, levelZ2);
    degree2 = degree;


    command_out(0x80);



    /* FORCE SENSOR OPERATION */

    /* Read request force sensor 1 */
    SingleTact = 0x0A;
    RRQST_reg = 132;
    RRQST_n = 2;
    ReadRQ_SingleTact();

    /* Read force sensor */
    Read_SingleTact();
    output1 = ((data_MSB << 8) + data_LSB);
    weight1 = ((float) output1 - 0x100) * 22 / 0x1FF;

    /* Read request force sensor 2 */
    SingleTact = 0x0C;
    ReadRQ_SingleTact();

    /* Read force sensor */
    Read_SingleTact();
    output2 = ((data_MSB << 8) + data_LSB);
    weight2 = ((float) output2 - 0x100) * 22 / 0x1FF;
  }
}


void GPIODConfig(void)
{
  /* GPIOD peripheral clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

  /* Configure PD0 in input mode for pushbutton and calibration buttons */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_6
| GPIO_Pin_7 | GPIO_Pin_8;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  /* Configure PD1 in output mode for LED */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOD, &GPIO_InitStructure);
```

```
}

void TIM2Config(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  /* TIM2 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

  /* GPIOA peripheral clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

  /* Configure PA1 to alternate function 1 (TIM2_CH2) */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, & GPIO_InitStructure);

  /* Connect TIM2 channels to PA1-AF1 */
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource1, GPIO_AF_TIM2);

  /* Enable TIM2 global interrupt and set priority to 0 */
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Period = (TIM2 f)/(desired interrupt f)
            = (84 MHz)/(1/1 ms)
            = 84000
            = Auto Reload Value + 1
     when fHCLK = 168 MHz & APB1Prescaler = 4 */

  /* Configure Time Base for TIM2 */
  TIM_TimeBaseStructure.TIM_Period = 83999;
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

  /* Configure TIM2 for output trigger */
  TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);

  /* Enable TIM2 counter and IT */
  TIM_Cmd(TIM2, ENABLE);
  TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
}

void TIM5Config(void)
{
  /* TIM5 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);

  /* GPIOA peripheral clock enabled in TIM2Config */

  /* Configure PA2 to alternate function 2 (TIM5_CH3) for INPUT signal */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOA, & GPIO_InitStructure);
```

```
  /* Connect TIM5 channels to PA2-AF2 */
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_TIM5);

  /* Enable TIM5 global interrupt and set priority to 0 */
  NVIC_InitStructure.NVIC_IRQChannel = TIM5_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Period = (TIM5 f)/(desired interrupt f)
            = (84 MHz)/(1/1 ms)
            = 84,000
            = Auto Reload Value + 1
  when fHCLK = 168 MHz & APB1Prescaler = 4 */

  /* Configure Time Base for TIM5 */
  TIM_TimeBaseStructure.TIM_Period = 83999;
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM5, &TIM_TimeBaseStructure);

  /* Configure TIM5 for output compare mode at Channel 3 */
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
  TIM_OC3Init(TIM5, &TIM_OCInitStructure);
  TIM_OC3PreloadConfig(TIM5, TIM_OCPreload_Disable);

  /* Enable TIM5 counter and IT */
  TIM_Cmd(TIM5, ENABLE);
  TIM_ITConfig(TIM5, TIM_IT_CC3, ENABLE);
}

void ADC1Config(void)
{
  /* GPIOC peripheral clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  /* ADC1 peripheral clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  /* Configure PC0 to analog input mode for ADC1 channel IN10 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* ADC common initalization
     single ADC mode
     no DMA access              */
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);

  /* ADC1 initialization
     12 bit conversion length
     regular channel conversion sequence (no scan mode)
     right aligned data */
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
```

```
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T2_CC2;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 2;
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular Channel 10 configuration (3 cycle sampling time) */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_3Cycles);
  ADC_RegularChannelConfig(ADC1, ADC_Channel_15, 2, ADC_SampleTime_3Cycles);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);

  /* Enable NVIC */
  NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Configure ADC1 interrupt to end of each conversion of a regular group */
  ADC_EOCOnEachRegularChannelCmd(ADC1, ENABLE);
  ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
}

void ADC2Config(void)
{
  /* GPIOC peripheral clock enabled in ADC1Config */

  /* ADC2 peripheral clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);

  /* Configure PC1 to analog input mode for ADC2 channel IN11 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_4;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* ADC common initalization
     single ADC mode
     no DMA access              */
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);

  /* ADC2 initialization
     12 bit conversion length
     regular channel conversion sequence (no scan mode)
     right aligned data */
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T2_CC2;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 2;
  ADC_Init(ADC2, &ADC_InitStructure);

  /* ADC2 regular Channel 11 configuration (3 cycle sampling time) */
  ADC_RegularChannelConfig(ADC2, ADC_Channel_11, 1, ADC_SampleTime_3Cycles);
```

```
  ADC_RegularChannelConfig(ADC2, ADC_Channel_14, 2, ADC_SampleTime_3Cycles);

  /* Enable ADC2 */
  ADC_Cmd(ADC2, ENABLE);

  /* Enable NVIC */
  NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  /* Configure ADC2 interrupt to end of each conversion of a regular group */
  ADC_EOCOnEachRegularChannelCmd(ADC1, ENABLE);
  ADC_ITConfig(ADC2, ADC_IT_EOC, ENABLE);
}

void ADC3Config(void)
{
  /* GPIOC peripheral clock enabled in ADC1Config */

  /* ADC3 peripheral clock enable */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC3, ENABLE);

  /* Configure PC2 to analog input mode for ADC3 channel IN12 */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* ADC common initalization
     single ADC mode
     no DMA access              */
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);

  /* ADC3 initialization
     12 bit conversion length
     regular channel conversion sequence (no scan mode)
     right aligned data */
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T2_CC2;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 2;
  ADC_Init(ADC3, &ADC_InitStructure);

  /* ADC3 regular Channel 12 configuration (3 cycle sampling time) */
  ADC_RegularChannelConfig(ADC3, ADC_Channel_12, 1, ADC_SampleTime_3Cycles);
  ADC_RegularChannelConfig(ADC3, ADC_Channel_13, 2, ADC_SampleTime_3Cycles);

  /* Enable ADC3 */
  ADC_Cmd(ADC3, ENABLE);

  /* Enable NVIC */
  NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
```

```
  NVIC_Init(&NVIC_InitStructure);

  /* Configure ADC3 interrupt to end of each conversion of a regular group */
  ADC_EOCOnEachRegularChannelCmd(ADC1, ENABLE);
  ADC_ITConfig(ADC3, ADC_IT_EOC, ENABLE);
}

/* Allows global variables to be visible using Live Watch */
void command_out(int c)
{
  GPIOB->BSRRH = GPIO_Pin_1; //set pin RS low
  GPIOD->ODR = (c << 8);
}

void I2C1Config(void)
{
  /* I2C1 peripheral clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);

  /* GPIOB peripheral clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  /* Configure PB6 and PB7 to alternate function 4 for I2C1 SCL and SDA */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /* Configure PB6 & PB7 to I2C1 alternate function (AF4) */
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_I2C1);
  GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_I2C1);

  /* Configure I2C1
     100 kHz SCL speed
     50% duty cycle
     7-bit slave address mode */
  I2C_InitStructure.I2C_ClockSpeed = 100000;
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1 = 0x00;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
  I2C_Init(I2C1, &I2C_InitStructure);

  /* Enable I2C1 */
  I2C_Cmd(I2C1, ENABLE);
}

void AccelCalib(void)
{
  if((((GPIOD->IDR) >> 3) & 0x01 != 0)
  {
    if(avgX > maxX)
    {
      maxX = avgX;
    }
  }

  if((((GPIOD->IDR) >> 4) & 0x01 != 0)
  {
    if(avgY > maxY)
    {
      maxY = avgY;
    }
```

```
  }

  if((((GPIOD->IDR) >> 5) & 0x01 != 0)
  {
    if(avgZ > maxZ)
    {
      maxZ = avgZ;
    }
  }

  if((((GPIOD->IDR) >> 6) & 0x01 != 0)
  {
    if(avgX < minX)
    {
      minX = avgX;
    }
  }

  if((((GPIOD->IDR) >> 7) & 0x01 != 0)
  {
    if(avgY < minY)
    {
      minY = avgY;
    }
  }

  if((((GPIOD->IDR) >> 8) & 0x01 != 0)
  {
    if(avgZ < minZ)
    {
      minZ = avgZ;
    }
  }
}

float AngleCalc(int levelX[1000], int levelY[1000], int levelZ[1000])
{
  /* Averagers for x, y, and z accelerometer conversions */
    for(int a=0; a<1000; a++)
    {
      sumX += levelX[a];
    }

    avgX = sumX/1000;
    sumX = 0;

    for(int b=0; b<1000; b++)
    {
      sumY += levelY[b];
    }

    avgY = sumY/1000;
    sumY = 0;

    for(int c=0; c<1000; c++)
    {
      sumZ += levelZ[c];
    }

    avgZ = sumZ/1000;
    sumZ = 0;

    if((((GPIOD->IDR) >> 3) & 0x3F) != 0) AccelCalib();

    rangeX = maxX - minX;
    rangeY = maxY - minY;
```

```
    rangeZ = maxZ - minZ;

    zeroX = (float) minX + (0.5 * rangeX);
    zeroY = (float) minY + (0.5 * rangeY);
    zeroZ = (float) minZ + (0.5 * rangeZ);

    /* Normalizes and centers about zero the conversion values */
    valX = (((float) avgX) - ((float) zeroX)) * (100 / (maxX - zeroX));
    valY = (((float) avgY) - ((float) zeroY)) * (100 / (maxY - zeroY));
    valZ = (((float) avgZ) - ((float) zeroZ)) * (100 / (maxZ - zeroZ));

    /* Transforms three-basis x,y,z vectors to two-basis x,(y+z) vectors*/
    x2 = valX * valX;
    y2 = valY * valY;
    z2 = valZ * valZ;
    result = sqrt(y2 + z2);

    if (valZ < 0) result = result * -1;
    ratio = valX / result;

    /* Finding angle and accounting for arctan limitations */
    if(valX>0 && result<0)
      angle = 3.141592654 + atan(ratio);
    else if(valX<0 && result<0)
      angle = atan(ratio) - 3.141592654;
    else
      angle = atan(ratio);

    degree = angle * 180 / 3.141592654;

    return degree;
}

/* 12C write operation for Single Tact force sensor (pg 11) */
void Write_SingleTact(void)
{
  /* wait until I2C1 is not busy anymore */
  while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

  /* Send start sequence */
  I2C_GenerateSTART(I2C1, ENABLE);
  /* Wait for EV5 (slave start acknowledge) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send SingleTact slave address + R/W bit (write = 0) */
  I2C_Send7bitAddress(I2C1, SingleTact, I2C_Direction_Transmitter);
  /* Wait for EV6 (slave address acknowledge) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send 0x02 to sensor (byte 0 of the Write Operation Data Packet) */
  I2C_SendData(I2C1, 0x02);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send register block offset (byte 1 of the Write Operation Data Packet) */
  I2C_SendData(I2C1, WST_reg);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send number of bytes to write, max 28 (byte 2 of the WODP) */
  I2C_SendData(I2C1, WST_n);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
```

```
  /* Send data to write (byte 3 to (N-1) of the WODP) */
  I2C_SendData(I2C1, WST_data);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send end of packet delimiter (byte N of the WODP) */
  I2C_SendData(I2C1, 0xFF);

  /* Send stop sequence */
  I2C_GenerateSTOP(I2C1, ENABLE);
}

/* 12C read request operation for Single Tact force sensor (pg 12) */
void ReadRQ_SingleTact(void)
{
  /* wait until I2C1 is not busy anymore */
  while(I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

  /* Send start sequence */
  I2C_GenerateSTART(I2C1, ENABLE);
  /* Wait for EV5 (slave start acknowledge) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send SingleTact slave address + R/W bit (write = 0) */
  I2C_Send7bitAddress(I2C1, SingleTact, I2C_Direction_Transmitter);
  /* Wait for EV6 (slave address acknowledge) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send 0x01 to sensor (byte 0 of the Read Request Operation Data Packet) */
  I2C_SendData(I2C1, 0x01);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send register block offset (byte 1 of the RRODP) */
  I2C_SendData(I2C1, RRQST_reg);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send number of bytes to read, max 32 (byte 2 of the RRODP) */
  I2C_SendData(I2C1, RRQST_n);

  /* Wait for EV8 (data has been written to data register and shifted out */
  while(!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
  /* Send end of packet delimiter (byte 3 of the RRODP) */
  I2C_SendData(I2C1, 0xFF);

  /* Send stop sequence */
  I2C_GenerateSTOP(I2C1, ENABLE);
}

void Read_SingleTact(void)
{
  /* Send second start sequence */
  I2C_GenerateSTART(I2C1, ENABLE);
  /* Wait for EV5 (slave start acknowledge) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_MODE_SELECT));

  /* Send SingleTact slave address + R/W bit (read = 1) */
  I2C_Send7bitAddress(I2C1, SingleTact, I2C_Direction_Receiver);
  /* Wait for EV6 (slave address acknowledge) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

  /* Enable master ACK */
  I2C_AcknowledgeConfig(I2C1, ENABLE);
```

```
  /* Wait for EV7 (ready to read data received from the slave) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));

  /* Copy latest byte read from I2C1 to temp_uh */
  data_MSB = I2C_ReceiveData(I2C1);

  /* Disable master ACK */
  I2C_AcknowledgeConfig(I2C1, DISABLE);

  /* Wait for EV7 (ready to read data received from the slave) */
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));

  /* Copy latest byte read from I2C1 to temp_lh */
  data_LSB = I2C_ReceiveData(I2C1);

  /* Send stop sequence */
  I2C_GenerateSTOP(I2C1, ENABLE);
}
```

11.1.2 Final ISR

```
/* Box Trainer ISR */

/* Header Files */
#include "stm32f4xx_it.h"

/* Definitions */
int levelX1[1000], levelY1[1000], levelZ1[1000], levelX2[1000], levelY2[1000], levelZ2[1000];
int i=0, j=0, k=0, l=0, m=0, n=0;
uint16_t ms_counter = 0;
uint8_t sec_counter = 0;
uint8_t min_counter = 0;
int flag = 0, adc1cnt = 0, adc2cnt = 0, adc3cnt = 0;
uint16_t start_time_ms = 0;
uint8_t start_time_sec = 0;
uint8_t start_time_min = 0;
uint16_t end_time_ms = 0;
uint8_t end_time_sec = 0;
uint8_t end_time_min = 0;
uint16_t elapsed_time_ms = 0;
uint8_t elapsed_time_sec = 0;
uint8_t elapsed_time_min = 0;

void TIM2_IRQHandler(void)
{
  /* Check TIM2 capture/compare register interrupt bit status */
  if(TIM_GetITStatus(TIM2, TIM_IT_CC2) == SET)
  {
    /* Clear TIM2's CCR interrupt bit */
    TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);

    /* Enables ADC1 software start conversion of regular channels */
    ADC_SoftwareStartConv(ADC1);

    /* Enables ADC2 software start conversion of regular channels */
    ADC_SoftwareStartConv(ADC2);

    /* Enables ADC3 software start conversion of regular channels */
    ADC_SoftwareStartConv(ADC3);
  }
}

void ADC_IRQHandler(void)
{
  /* Check ADC1 end-of-conversion interrupt flag */
```

```
if(ADC_GetITStatus(ADC1, ADC_IT_EOC) == SET)
{
  /* Clear interrupt flag */
  ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);

  if(adc1cnt%2 == 0)
  {
    /* Copy ADC conversion to level */
    levelX1[i] = ADC_GetConversionValue(ADC1);

    i++;
    if(i==1000) i=0;
  }

  else
  {
    /* Copy ADC conversion to level */
    levelX2[l] = ADC_GetConversionValue(ADC1);

    l++;
    if(l==1000) l=0;
  }

  adc1cnt++;
}

/* Check ADC2 end-of-conversion interrupt flag */
if(ADC_GetITStatus(ADC2, ADC_IT_EOC) == SET)
{
  /* Clear interrupt flag */
  ADC_ClearITPendingBit(ADC2, ADC_IT_EOC);

  if(adc2cnt%2 == 0)
  {
    /* Copy ADC conversion to level */
    levelY1[j] = ADC_GetConversionValue(ADC2);

    j++;
    if(j==1000) j=0;
  }

  else
  {
    /* Copy ADC conversion to level */
    levelY2[m] = ADC_GetConversionValue(ADC2);

    m++;
    if(m==1000) m=0;
  }
}

/* Check ADC3 end-of-conversion interrupt flag */
if(ADC_GetITStatus(ADC3, ADC_IT_EOC) == SET)
{
  /* Clear interrupt flag */
  ADC_ClearITPendingBit(ADC3, ADC_IT_EOC);

  if(adc3cnt%2 == 0)
  {
    /* Copy ADC conversion to level */
    levelZ1[k] = ADC_GetConversionValue(ADC3);

    k++;
    if(k==1000) k=0;
  }
```

```
    else
    {
      /* Copy ADC conversion to level */
      levelZ2[n] = ADC_GetConversionValue(ADC3);

      n++;
      if(n==1000) n=0;
    }
  }
}



void TIM5_IRQHandler(void)
{
  if(TIM_GetITStatus(TIM5, TIM_IT_CC3) != RESET)
  {
    TIM_ClearITPendingBit(TIM5, TIM_IT_CC3);

    ms_counter++;

    if(ms_counter == 1000)
    {
      sec_counter++;
      ms_counter = 0;
    }

    if(sec_counter == 60)
    {
      min_counter=min_counter+1;
      sec_counter = 0;
    }

    if((GPIOD->IDR & 0x0001) && (flag == 0))
    {
       start_time_ms = ms_counter;
       start_time_sec = sec_counter;
       start_time_min = min_counter;

       GPIOD->BSRRL = GPIO_Pin_1;
       flag = 1;
     }

     if((!(GPIOD->IDR & 0x0001)) && (flag == 1))
     {
       flag = 2;
     }

     if((GPIOD->IDR & 0x0001) && (flag == 2))
     {
       end_time_ms = ms_counter;
       end_time_sec = sec_counter;
       end_time_min = min_counter;

       GPIOD->BSRRH = GPIO_Pin_1;
       flag = 3;
     }

     if((!(GPIOD->IDR & 0x0001)) && (flag == 3))
     {
       flag = 0;
     }

    if(end_time_ms > start_time_ms)
    {
      elapsed_time_ms = end_time_ms - start_time_ms;
```

```
    }

    else if (end_time_ms < start_time_ms)
    {
      elapsed_time_ms = (1000 - start_time_ms) + end_time_ms;
    }

    else
    {
      elapsed_time_ms = 0;
    }

    if(end_time_sec > start_time_sec)
    {
      elapsed_time_sec = end_time_sec - start_time_sec;
    }

    else if (end_time_sec < start_time_sec)
    {
      elapsed_time_sec = (1000 - start_time_sec) + end_time_sec;
    }

    else
    {
      elapsed_time_sec = 0;
    }

    if(end_time_min > start_time_min)
    {
      elapsed_time_min = end_time_min - start_time_min;
    }

    else if (end_time_min < start_time_min)
    {
      elapsed_time_min = (1000 - start_time_min) + end_time_min;
    }

    else
    {
      elapsed_time_min = 0;
    }
  }
}
```

12.0 Appendix: Data Sheets