Autonomous Underwater Robots

Department of Electrical and Computer Engineering
Bradley University, Peoria, IL

Ryan Lipski, Cameron Putz, and Nicholas Sikkema
Advisor: Joseph A. Driscoll, Ph.D.

April 30, 2015

Abstract

The goal of this project was to build a swarm of autonomous robots to map underwater terrain. Research was conducted by the Autonomous Underwater Robots team to determine the best means in which to approach this problem. Specialized detection methods were generated by using blue LEDs and blue filtered photodiodes. The physical design of the robots involved using RC submarine platforms that were modified to include additional subsystems. The additional subsystems used for each submarine are the detection array, the power system, the motor control system, and the camera system. Individual swarm members were designed to swarm using minimalistic swarming techniques. They follow cohesion, alignment, and separation criteria in order to complete the task successfully. Image stitching software was used to compile an image from the smaller images taken by each of the swarm members. The swarm members will also be designed to meet specific cost criterion. Our proposed project can be accomplished with less than $1000. The team investigated various cost-effective methods to waterproof the submarines. The autonomous submarines were designed with societal and environmental impacts taken into consideration.

Table of Contents

Abstract i

I. Introduction and Overview 1

A. Problem Background 1

B. Problem Statement 1

C. Constraints of the Solution 2

II. Statement of Work 2

A. Functional Requirements 2

B. Nonfunctional Requirements 3

C. Design Overview 3

1. System Block Diagram 3

2. Subsystems 4

3. State Diagram 9

4. Division of Labor 9

5. Software 10

4. Hardware 11

III. Design Testing and Validation 14

IV. Conclusions 15

V. Acknowledgements 15

VI. References 16

VI. Appendix 17

(A) Parts List 17

(B) Glossary 18

(C) Detailed cost analysis 19

(D) Detailed schematics, models, flowcharts 20

(E) Detailed experimental results 24

(f) Theory of operation 25

(G) Video, web links 26

(H) Code and Documentation 26

Main.c 26

Master_Build_Control.h 27

Main_Program.c 29

Main_Program.h 30

Motor_Control.c 30
Motor_Control.h 33
Swarming.c 34
Swarming.h 41
State_Machine.c 41
State_Machine.h 48
ADC.c 49
ADC.h 50
CD4051.c 51
CD4051.h 52
I2C_Interface.c 53
I2C_Interface.h 55
TWI_Master.c 55
TWI_Master.h 60
USART.c 63
USART.h 67
Interrupt.c 67
Interrupt.h 68
Timer.c 68
Timer.h 70
DRV8830.c 71
DRV8830.h 84
LSM303.c 85
LSM303.h 93
CLS15.c 93
CLS15.h 98
XPEBBL.c 98
XPEBBL.h 100
Battery.c 100
Battery.h 101
MPX5010GP.c 101
MPX5010GP.h 102
Test_Battery.c 103

Test_Battery.h 104
Test_H_Bridge.c 104
Test_H_Bridge.h 109
Test_I2C_Compass.c 109
Test_I2C_Compass.h 110
Test_LED.c 110
Test_LED.h 111
Test_Motor_Control.c 112
Test_Motor_Control.h 115
Test_Multiplexer.c 115
Test_Multiplexer.h 118
Test_Photodiodes.c 118
Test_Photodiodes.h 122
Test_Pressure_Sensor.c 123
Test_Pressure_Sensor.h 124
Test_Swarming.c 124
Test_Swarming.h 125
Code Documentation 125
(I) Datasheets 125
(J) Detailed Distribution of Tasks 125
(K) Recommendations for Future Work 127

I. Introduction and Overview

Over the years, people have looked in awe at the large size of the oceans that cover our planet. Oceans cover more than 70% of the surface of the Earth; however, only 5% of this has been seen by human eyes [1]. Oceans hold vital information for various scientific communities. Biologists can gain information about the fauna and flora that survive in these conditions [2]. This information includes the patterns of animal migration through the oceans and even the health of the coral that occupy the reefs. Geologists can understand the tectonics of the planet, through the movement of the plates that make up the underwater terrain [2]. Obtaining images that hold this vital information is where this project comes into play.

A. Problem Background

This project can be summarized by developing a swarm of autonomous underwater vehicles (AUVs) to map underwater terrain. Since 1957, AUVs have been exploring bodies of water without human interaction. AUVs have since become a much more useful and feasible idea. AUVs operate for 8 to 10 times less cost and operate for longer periods when compared to ship-based surveys [3]. Many of the AUVs available commercially today have been designed for ocean use. The AUVs are typically equipped with powerful sonar sensors that allow them to detect objects at depths of 9000 meters [4]. The cost of these AUVs can vary greatly depending on the imaging ability and pressure rating. Purchasing commercial AUVs to complete this project is not possible due to the financial constraints.

Swarming is the collective behavior of individuals that allows them to move as a group. One paper on AUV swarming discussed using a swarm to map an underwater minefield [5]. The paper discussed an ideal swarm size as well as a simulation they conducted. The group found multiple papers discussing an AUV swarming project in Europe called Cocoro, which stands for collective cognitive robots [6]. The project has successfully implemented an organized swarm of AUVs. This project is similar to our proposed project in that they both use small, inexpensive submarines with minimal inter-robot communication.

B. Problem Statement

Through discussion with the project advisor, the objectives, functions, and constraints were created. The goals of this project are summarized by five primary objectives and seven primary functions. The objectives include cost minimization, autonomous motion, durability, underwater mobility, and portability. Many of these objectives were set in accordance to guidelines set by Bohm and Jenson [7]. Their book on underwater robotics was used to understand the necessary aspects for creating an underwater robot. The functions include detecting other submarines, taking images of underwater terrain, maneuvering through a body of water, having a battery life of at least 15 minutes, operating as a swarm when near each other, creating an image from multiple images, and surfacing upon the battery level dropping below 5%. Multiple constraints affected the design. The first constraint is to avoid harming underwater organisms. To accomplish this, the swarm must not pollute the water or collide with obstacles. The second constraint is battery life. Each submarine shall have enough battery life to such that the swarm can map the entire pool at the Markin Center. A third constraint is that the submarines must have neutral buoyancy. The submarines will be constructed and sealed appropriately to withstand the pressures that exist at that depth. The sail must not leak. The sail must be constructed so that the electronics in the sail must not get wet. The last constraint is that the electronics must fit inside of the platform.

C. Constraints of the Solution

TABLE I
CONSTRAINTS OF THE SOLUTION

Constraints
Must not harm underwater life forms
Must have enough battery life to map a reasonably sized area
Must have neutral buoyancy
Must withstand the water pressure
Sail must not leak
Electronics must fit on the platform

II. Statement of Work

The research shown below goes over various items that are needed to complete this project. The items that are needed for this project include detecting other submarines, AUV research, and some details on swarming techniques.

A. Functional Requirements

TABLE II
FUNCTIONAL REQUIREMENTS FOR THE OVERALL SYSTEM AND SUBSYSTEMS

Functional Requirement	Specification/Method
Each submarine shall detect other submarines	1.22 meters, on PCB's
The swarm shall take images of underwater terrain	640x480, compiled image
The swarm shall maneuver through a body of water.	Depth: ± 10 cm, up to 60cm, constant speed ± 5%, I^2C controlled
The submarines shall have a battery life of at least 15 minutes	Battery life: 25 min
The swarm members shall operate as a swarm when near each other.	Passive sensing while close
Software shall create an image that is collected from the individual submarine images.	10% image gaps or less
Each submarine shall surface upon its battery level dropping below 5%.	Surface at 10% battery life

B. Nonfunctional Requirements

TABLE III
NONFUNCTIONAL REQUIREMENTS FOR THE OVERALL SYSTEM AND SUBSYSTEMS

Objective	Metric
Minimize Cost	Determined by the production cost of one member of the swarm.
Autonomous	The amount of human interaction needed for the swarm to function.
Mobile Underwater	The turn radius of each swarm member and how close the swarm member is to neutrally buoyant.
Durable	The amount of places that fail per swarm member.
Portability	The perceived size and weight of each swarm member.

C. Design Overview

1. System Block Diagram

The system black box, shown in Fig. 1, describes the final system in terms of inputs and outputs. Fig. 2 shows the entire system hardware and that the system is powered by four nickel–metal hydride (NiMH) batteries with connection to a voltage step up regulator. The microcontroller is an ATMega328P. The team used a multiplexer to connect seven devices (five photodiodes, the battery voltage, and the pressure sensor) through one analog-to-digital converter (ADC). This reduced the amount of ADCs that the system needed from seven to one, which allowed the ATMega328P to be used. The ATmega328P also has an inter-integrated circuit (I^2C) bus that connects three h-bridges and an inertial measurement unit (IMU). The h-bridges individually drive the DC brushed motors. The last subsystem that the ATMega328P controls is the light-emitting diode (LED) driver circuit where three digital I/O pins were used to control the LED intensity using pulse width modulation (PWM). These systems will be discussed in depth below.

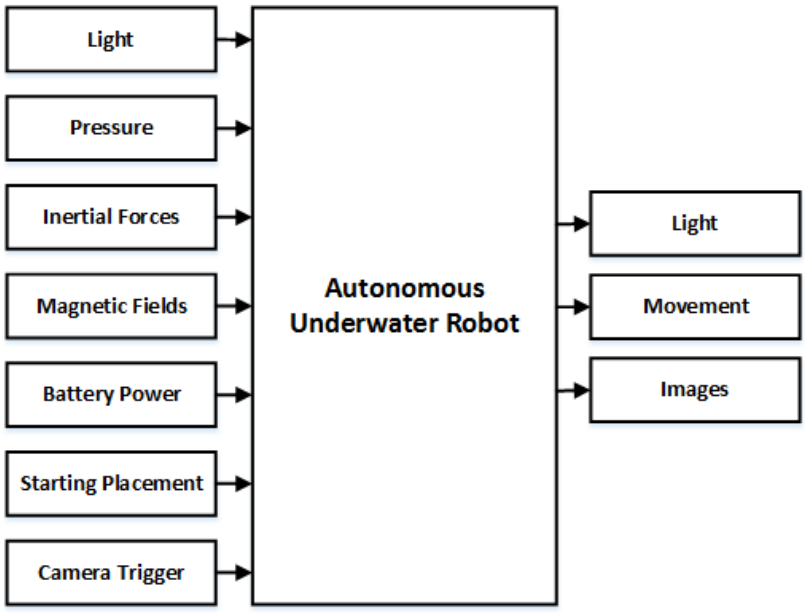

Fig. 1. System Black Box. The overview of each robot's inputs and outputs is shown.

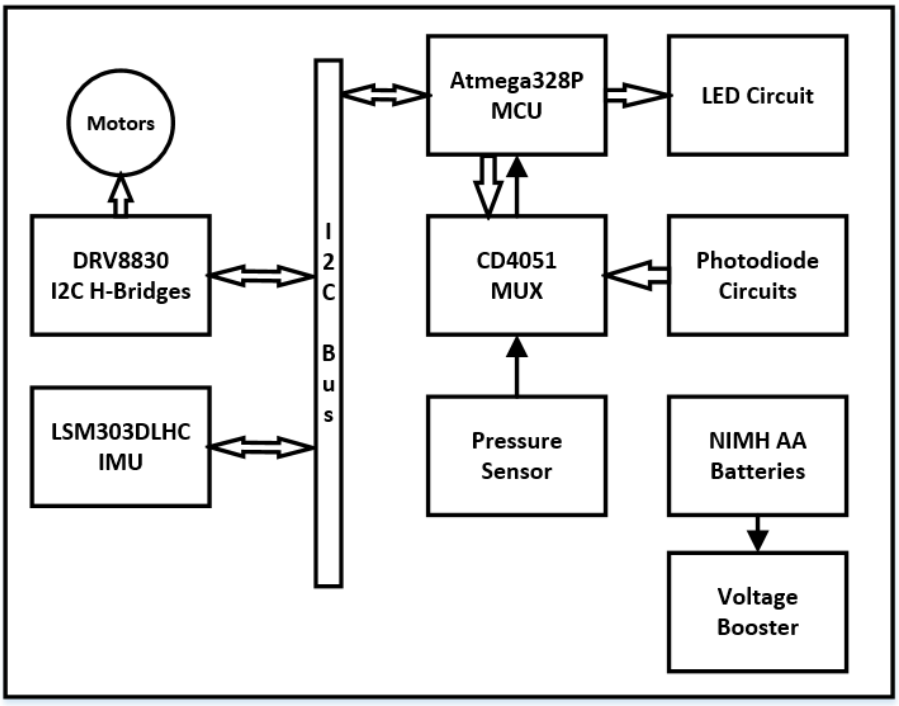

Fig. 2. System block diagram. The entire hardware system is shown.

2. Subsystems

a. Detection Array

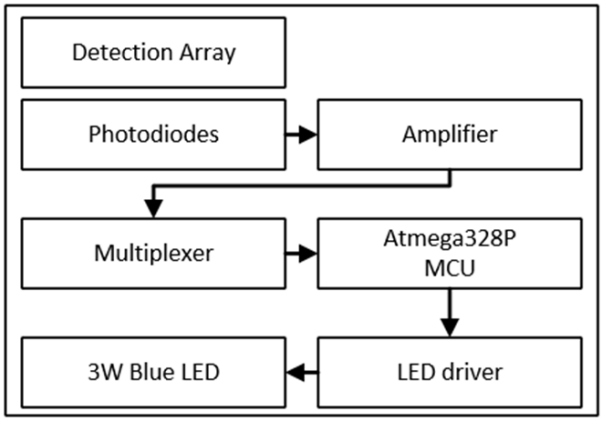

Fig. 3. Detection array block diagram.

The detection array system is a key system to developing swarming robots. The Autonomous Underwater Robotics group looked into several detection methods for this project. These methods include computer vision, electromagnetic sensing, acoustic sensing, and using high-powered low frequency wireless. None of these sensing techniques worked for all of the objectives and constraints. The team needed an inexpensive, reliable, low-power way to detect other submarines underwater. The system the team decided to go with uses the visible light spectrum to detect the other submarines using LEDs and photodiodes. The LEDs display the submarines' current location to adjacent swarm members and the photodiodes receive that light. A swarm member will then determine direction and distance based on the

light intensity and angle. The color of LED that was chosen was crucial to getting the distance that was needed. This color is a blue, around the 480 nm range. This wavelength has the lowest attenuation coefficient of the electromagnetic spectrum as shown in Appendix F.

TABLE IV
PHOTODIODE COMPARISON

	Osram photodiode	**Everlight photodiode**
Saturation	91.44 cm – 4.9 Volts	10.16 cm – 4.89 Volts
Max Distance	457.2 cm – 0.6 Volts	335.28 cm -- 0.120 Volts
Linearity	Linear	Non-Linear
Ambient Light	100% Saturation	29% Saturation
Price	$8.85	$0.59

The purpose of the detection array was to gather information that would allow the submarines to avoid obstacles and operate as a swarm when the submarines were close together. The front photodiode and LED are used for active sensing, where the LEDs light reflect off an obstacle and is received by the front photodiode. The two photodiodes and LED on each side a submarine's sail were used for passive sensing. The side LEDs transmits light that adjacent submarines detect with their side photodiodes. The two photodiodes on each side are separated by a baffle. The baffle is used to shade one of the photodiodes if the incoming light is not directly hitting the side of the sail. This baffle and photodiode setup is used to gather separation, cohesion, and pseudo-alignment data, the three criteria of the minimalistic swarming algorithm. As seen in Fig. 4, each submarines has twenty detection zones. The front zone is used only for obstacle detection. The minimalistic swarming algorithm was implemented by placing detected light into one of the eighteen passive sensing zones. The algorithm alters variables that are weighted depending on which zone the adjacent submarine(s) was detected in. Bench-top testing was done with the detection array hardware and the minimalistic swarming algorithm. Another detection array was used to transmit the blue light to simulate an adjacent submarine. An adjacent submarine could be detected in any one of the eighteen passive sensing zones, depending on its location.

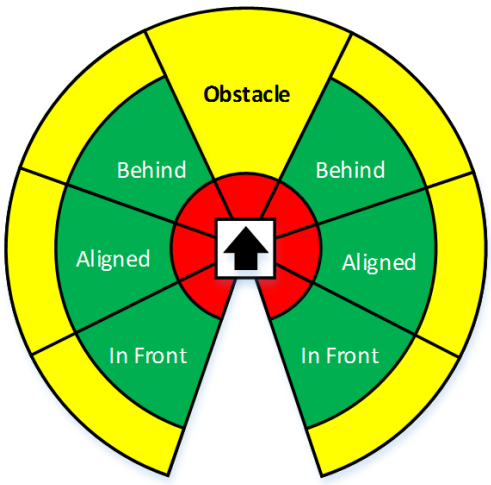

Fig. 4 Photodiode detection zones. Green zones are ideal locations for adjacent swarm members.

b. Camera System

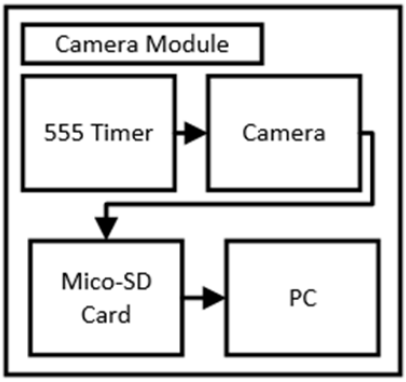

Fig. 5. Camera system block diagram.

The camera system was designed to be independent of the other systems and to take pictures autonomously. The team decided to go with an 808 pinhole camera as it was easy to interface with. The camera was originally designed as a key fob and was used for taking images discreetly. The camera circuit was designed to be small and fit seamlessly onto a key ring. Because of the circuit's small size, it was ideal for our system and was easier to waterproof. The camera had a 3.7 V battery and used a push button switch to take pictures. The camera also used a micro-SD to store the images. After the submarines have finished taking the pictures, the micro-SD cards are taken to a computer. On the computer, there are two tasks that need to be done: time-stamp removal and photo merging. Using Photoshop (Adobe Systems, San Jose, California), the timestamp is removed. Photoshop has a batch feature that records commands and applies them onto all of the pictures in a folder. The team used this feature to remove area taken up by the timestamp and then blend the empty area with its surroundings. The blend feature of Photoshop helped later on with the photo merging by having some objects to work with instead of having a black square. Continuing to use Photoshop, the team took the pictures after timestamp removal and used the collage feature of photo merging. This feature outputs a flat image by rotating and shifting the input images.

c. Motor Control

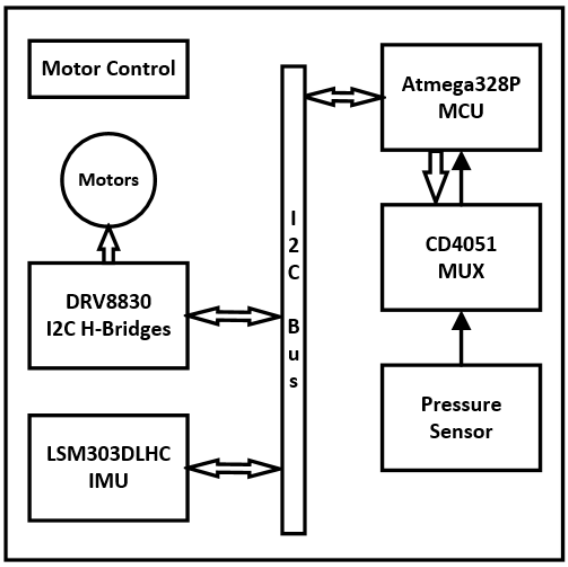

Fig. 6. Motor control block diagram.

The motor control subsystem is comprised of the Atmega328p microcontroller, the multiplexer, the pressure sensor, the h-bridges, the IMU, and the DC motors. The h-bridges and IMU are connected to the microcontroller through I^2C, which minimized the amount of pins required to interface this subsystem. Normally with h-bridges there is a constant need to regulate the h-bridges for an expected direction and speed. Constantly having to regulate the h-bridges would require several more pins to be used. An additional benefit of these h-bridges is that they handle the faults and regulate the output voltage. Should the motor suddenly pull more current than expected, the h-bridges reduce the PWM percentage to reduce the power being drawn. If an overcurrent occurs for an extended period, a fault is triggered.

Having three h-bridges and three DC motors, resulted in three control systems. The x-axis (turning) and y-axis (forward/reverse) motors along with two h-bridges were implemented using proportional-integral-derivative (PID) speed control. The IMU was used for feedback. The IMU returned acceleration values, which were then integrated to provide velocity values. The z-axis (diving/surfacing) motor and an h-bridge were implemented using PID position control. The pressure sensor was used for feedback. The pressure sensor returned PSI values that were then converted into a depth values.

The force of gravity on the IMU was a source of error in the speed control algorithm. On the Earth's surface, there is a constant 9.8 m/s^2 acceleration on all objects. In order to remove the effect of gravity on the accelerometer one needs to know the tilt of the device. The IMU the team used was designed for relative compass orientation using a magnetometer and accelerometer. In order for the team to have an accurate acceleration measurement, the team need would have needed a gyroscope. This would allow the team to identify the tilt of the IMU and compensate for gravity. Knowing that the submarines did not have the ability to accurately measure the velocity with the IMU, the team switched the motor speed control to solely receive information from the swarming algorithm.

The z-axis motor, an h-bridge, and the pressure sensor were implemented using PID position control. In order to save power, a ± 3 cm dead band was added to the algorithm. The dead band prevents the submarine from constantly over correcting itself. With the dead band in place, submarines in the swam

could have been as much as 6 cm apart in depth. This difference in depth was not an issue. The sensing capabilities of each of the submarines were able to detect each other even with the depth discrepancy.

d. Power System

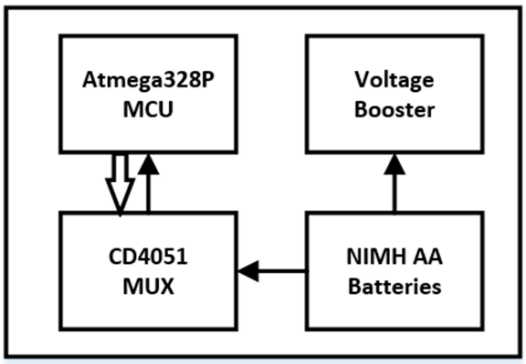

Fig. 7. Power system block diagram.

The power subsystem is comprised of the NiMH batteries, the voltage booster, the Atmega328p microcontroller, and the multiplexer. The NiMH batteries are the main power source of the submarine. These batteries power every component, except for the camera system, which has its own battery supply.

The NiMH batteries used were AA sized with 1.34 V at full charge and a capacity of 2500 mAh. The batteries were connected in series, which provided different voltages depending on how many batteries were being used. Both three and four batteries were used in series to power the hardware, providing a 4.02 V and a 5.36 V supply. The 5.36 V supply was used to power the h-bridges, the LED drivers, and the pressure sensor. The 4.02 V supply was used to power the voltage booster, which then provided a stable output of 5.0 V. A diode was added in series with the voltage booster's output for reverse current protection. The forward voltage drop across the diode resulted in a 4.5 V output. This output was then used to power all of the other hardware. Additionally, the voltage booster was used in conjunction with three of the batteries and an ADC to determine when the system is below 10%. In this scenario, the submarine surfaces and goes to the edge of the testing pool. As seen in the Appendix F, when the batteries reach the remaining 20% of their charge, the voltage starts to suddenly drop. With the voltage booster, the submarine is able to continue operating until the battery drains to unsafe levels.

e. Submarine Construction

In order to add all of the new electronics to the Motorworks submarine platform, multiple modifications were made. The original electronics that were potted in the front of the submarine were removed. A clear plastic cylinder was, shown in Fig. 8, added to the front of the submarine to house the new electronics. The cylinder was cut in half to make the task of connecting all of the electronics inside the new sail easier. Hot glue was used to re-waterproof the cylinder before each in-water test. The bottom section of the cylinder was drilled to feed in the wiring from the three motors and the battery compartment. Epoxy was applied to the bottom portion of the sail to waterproof and strengthen the areas that were drilled. Hot glue was also added to all that other areas that were deemed susceptible to leaking.

The photodiodes and LEDs of the detection array were mounted by soldering them to a small section of header pins. This produced a much more rigid connection than directly soldering wires to the components. Tape was placed over the components of the detection array in order to prevent the LEDs from reflecting light inside the sail and skewing the photodiode readings. Female-to-female jumper wires were used for many of the electrical connections. Electrical tape was used to insulate the power and I^2C junctions

that were exposed inside the sail. With all the hardware connected and placed in the sail, there was very little extra space left over, but all of the components very able to function properly.

Fig. 8. One of the fully modified and constructed submarines.

3. State Diagram

To control the overall operation of the submarine the main state machine has three main portions: the initialization, the normal operation, and end of operations. For the initialization, the microcontroller initializes all of the subsystems. Once this is done the system then waits for the submarine to be in water before continuing. After the submarine is placed in the water, the system starts a calibration circle that is used to correct the offset of the op-amp circuit for the photodiode. Next in the state machine is the normal operation. During this portion, all of the submarine's subsystems are interacting with each other in order to act like a member of a swarm. When the system is running low on power, or if there are enough faults from the h-bridge, then the system goes into last portion of the state machine. In these states, the submarine surfaces and goes to the edge of the testing pool. These final states were added to ensure easy retrieval of the submarines. These states can be further seen in Fig. 9.

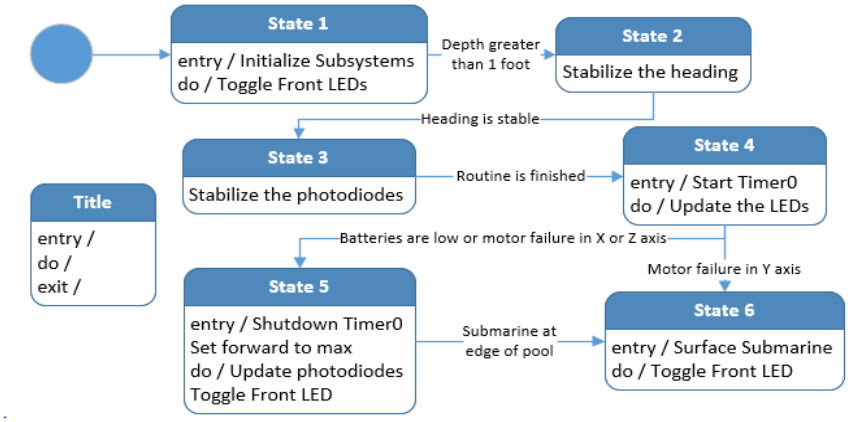

Fig. 9. State Diagram for the code

4. Division of Labor

The division of labor was evenly divided between the three members of the AUR group. Cameron was in charge of the hardware and focused on three subsystems: the camera system, the detection array, and the power system. For the camera system, Cameron designed all the hardware and completed the interface for the camera module. Cameron developed the hardware for the detection array. He developed and tested the circuits and created PCB's to interface reduce the size of the circuits. For the power system, Cameron focused on helping interface and develop the power distribution channels.

Ryan was involved in both the hardware and software design of the system. Ryan primarily worked on the detection array, the motor control system, and the power system. For the detection array, Ryan helped validate the final design and he developed the swarming software. For the motor control system, Ryan wrote the PID control software and conducted the depth tracking testing. Ryan also worked on the power system where he performed battery life testing and calculations.

Nick was involved in the software design of the system. Nick primarily worked on the motor control system, the power system, and the camera system. For the motor control, Nick wrote the I2C code to interface with the h-bridge. For the power system, Nick worked on the routine for when the battery is below 10%. For the camera system, Nick work on the batch photo editing for the photo merging. The complete division of labor is shown in the Appendix J.

5. Software

The design of the software was separated into three different portions: the main state machine, the backend code, and the testing code. Additionally, there were two ideas when designing the functions. First is to make the name understandable without actually reading the code. When looking at the function name the team wanted the people that were reading the code to understand the code's purpose. To go along with this, the larger functions were separated out into subfunctions. As seen in Fig. 10, the battery read voltage function uses the battery read ADC and the ADC convert to voltage functions. This improves readability and allows other students to understand the code.

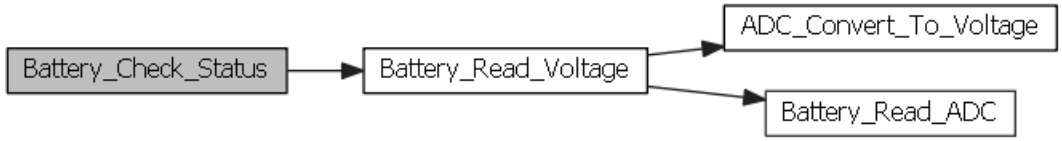

Fig. 10. Battery check call graph.

a. Software Testing

The testing code is based off unit testing. Instead of having rough code that needs to be commented out to test certain portions of the portions of the code, the code was designed to build only certain portions. This was done in order to test new revisions of the code, without having to modify the existing code for testing. This allowed the team to test the code, fix the bug, and then test high levels of the code.

b. Software Backend

The last portion of the code is the backend of the code. This portion consists of the code that is used to interface the microcontroller with the subsystems.

i. Motor Control

In order to control the h-bridge, an I^2C interface was created for the Atmel TWI Master that allows the team to reuse code instead of having to rewrite interfacing code several times. There are two significant portions of h-bridge code: the brown out protection and the fault system. For the brown out protection, it protected against two scenarios: more than one motor starting and the motor from jumping voltage too quickly. Adding these portions helped reduce the noise on the system ground. The fault system worked in conjunction with the state machine. Every time the function would check the fault register through I^2C. If it encountered a fault, it would restart the h-bridge and set flags to indicate that a motor has failed after a set amount of restarts. This indicates to the state machine that there is a system problem and sets the submarine to go to the edge of the pool.

PID control was implemented in software to control the motors. The h-bridge was not capable of handling negative control signals. When the control signal went negative, the control signal was made

positive and the motor direction was switched. The submarines were programmed to have a bias to move forwards, as this is required to complete the mapping. The outputs from the swarming algorithm factored into the motor control software to produce autonomous movements.

ii. Detection Array

The photodiode outputs of the detection array were read by selecting the associated multiplexer pin and the reading the ADC. During the swarming algorithm state of the state machine, all of the photodiode register values were updated. These photodiode values were used in software algorithm that sorted the incoming light on either side of the sail in a single zone. The resulting zone knowledge led to the altering of variables that factor into the motor control to produce intelligent, autonomous behavior.

To reduce the power consumption of the three watt LEDs, PWMs were implemented to reduce the on time. The PWM is controlled by a state machine. The LED is turned on for a small amount of time and then the next LED is turned on. Between each LED being on, there is a small transitional period where no LEDs are on.

iii. Power System

The power system is used in several portions of the project. One of the more important situations is the indication of low power. Connecting three of the four batteries to one of the ADCs, the team was able to determine when the battery voltage dropped below 10%.

c. Software State Machine

To combine all of the code that was written, a state machine was created. The group saw this as the best method due to the needed control between the stages. In order to accomplish this, flags were added to control which portions of the code were on and when they update.

d. Software Documentation

All of the project code was documented in Doxygen, making it easier for others (the team's advisor and other groups) to understand the code and show the progress that the group made. Using the call graphs feature of Doxygen, a block diagram that shows the function calls for a given function, the team was able to debug the code more effectively. Another benefit is that this documentation may be of some use to department projects in the future. If students need code for similar parts or if a group wants to continue this project they will be able to use the team's code effectively without needing to fully analyze the code.

4. Hardware

After choosing the type of detection method, the AUV team had to develop a system for using the LED's and photodiode combination repeatedly and consistently i.e. (develop an amplifier that outputs consistent values for the photodiodes and develop a method to get each LED to output at the same intensity). Using the components that were readily available in the laboratory, several design iterations were completed. These designs are shown in Appendix D and they were completed to get accurate output values from the photodiodes. The same was done for the LED driver design.

The photodiode circuit, after several iterations, was designed to be a transimpedance amplifier. This design was only made possible by using an operational amplifier that was not available in the lab. This amplifier had the required characteristics: low bias current and rail-to-rail voltage. The bias current for this op-amp is 3 pA. This is characteristic is shown in the datasheet in Appendix I. The op-amp needs low bias currents to permit normal photodiode operation.

The LED driver circuit was designed to be a current source. The driver design was accomplished by using basic electronics theory, and the LED circuit was developed in OrCad and simulated using PSpice, as shown in Appendix D. This circuit was then implemented on a surface mount board.

The AUR team developed two surface mount boards for the detection array. They were developed using Eagle 7.1.0. Originally, there was a single board for the detection array, however the detection array was divided into two boards to keep the high current (i.e. above 20 mA) isolated from the sensitive photodiode electronics. The design of these boards can be seen in Appendix D. To minimize the heat dissipation of the high current board, a copper ground plane was added to one side of the board. Due to the components and the size of the photodiode interface board, a copper layer was not added to the opposite side of the board. Once the printed circuit boards (PCBs) were developed, shown in Fig. 11, and the surface mount components were soldered to them, the team began testing the detection array.

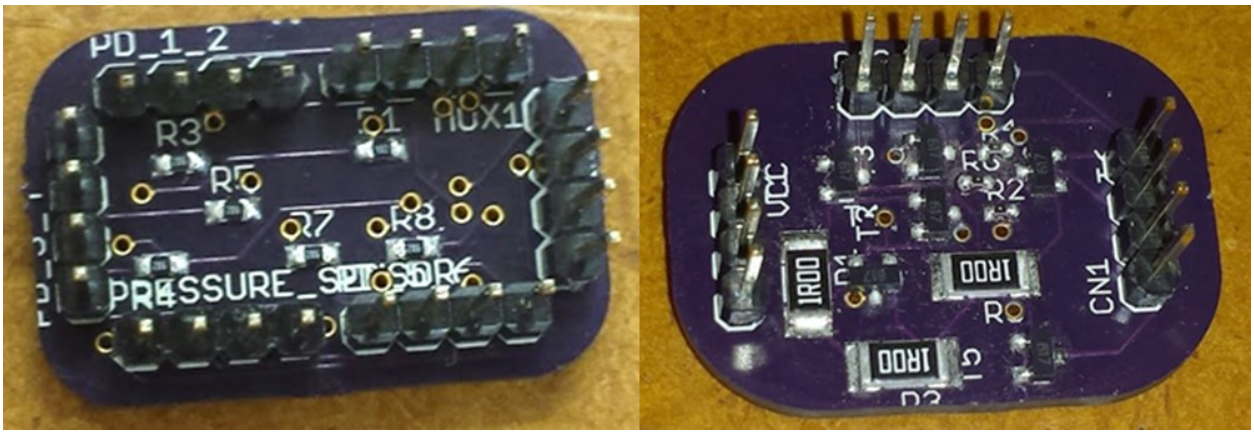

Fig. 11. Detection array PCB's. Photodiode board (left), LED driver (right).

A 555 timer circuit was connected in an astable oscillator configuration to give an approximate square wave with a duty cycle of 50%. The AUR team used the developed circuit and soldered it to a perfboard. The team then interfaced the perfboarded 555 timer circuit to the image capturing button of the camera by soldering jumper wires to them. The 555 timer circuit was designed to capture pictures every 1.5 seconds. This schematic is shown in Appendix D.

A PCB was designed to interface the three h-bridges to the microcontroller easily. The PCB was designed to include TI's PowerPad™, which allows the PCB to have adequate heat dissipation. The PowerPad was connected directly to the ground plain to further increase the effectiveness of the thermal dissipation. When the completed board was returned to the AUR team, there were a couple design flaws found. The first flaw was that the copper traces' widths, or copper thickness, was too small and could not adequately handle the current flowing through them. By adding multiple VCC connections to the board, it was able to divide the current among the three h-bridges. This was effective in fixing the first problem. The second problem was that there was too much noise on the I^2C pins. The noise affected the communications to the point where there was no communication between any of the h-bridges and the microcontroller. To fix this, the team added a 10 µF capacitor between the VCC and ground pins, which eliminated the noise and restored communications. These fixes along with the board are shown in Fig. 12.

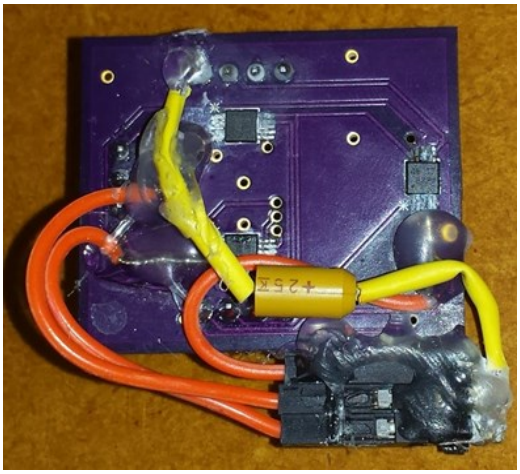

Fig. 12. Motor driver PCB. Motor driver PCB with the fixes to h-bridge VCC and I^2C.

d. Economic Analysis

As shown in table V, each of the four submarines ended up costing $173.71. The team estimated a cost of $300 per submarine, and the minimize cost metric is based off this. A cost breakdown can be seen below. The full cost breakdown can be seen in the Appendix C. The most expensive component for each submarine was the original submarine platform. The total cost of the swarm ended up being $694.83.

TABLE IV
BILL OF MATERIALS

Quantity	Description	Price
21	Resistors	$5.72
6	Transistor	$3.14
1	IMU	$9.95
1	4 pack AA Batteries	$5.99
3	I2C H-bridge	$7.32
1	Camera Module	$11.99
1	Pressure Sensor	$16.09
1	Motorworks Submarine	$61.99
3	Blue LED	$7.86
1	Step-Up Voltage Regulator	$3.95
1	Adafruit Pro Trinket	$9.95
1	Multiplexer	$0.44
2	Op-amp	$7.20
5	Photodiode	$1.90
1	PCB's	$6.13
1	Other	$14.09
	Total per submarine	**$173.71**
	Total for swarm	**$694.83**

III. Design Testing and Validation

The motor control testing was initially done out of water. Before any of the h-bridges were functional, the team had to correct some hardware and software issues. An issue with the I^2C communication was discovered when we began testing the h-bridges. Another issue was noise on the I^2C bus from the motors. As discussed earlier, this issue was fixed by adding a 10 µF capacitor to the h-bridge board. All three h-bridges were functional after these fixes. Temperature readings were taken from the h-bridge PCB and it was determined that heat would not be an issue.

Depth tracking tests shown in Fig. 13 proved that all the hardware and software was working together properly. Gains in the PID position control system were set to while testing on a bench-top. Pressurized air was applied to the pressure sensor in order to simulate the submarine being underwater. Different depths were simulated to see if the controller would react appropriately. The gains used during depth tracking testing were approximately P = 4.1, I = 4.1 and D = 2.1. The controller was tuned to be underdamped, as seen by the overshoot of a few centimeters when the submarine was pushed away from its set point. The benefit of underdamping was that the submarines had reduced rise times so they could reach the 60 cm depth faster.

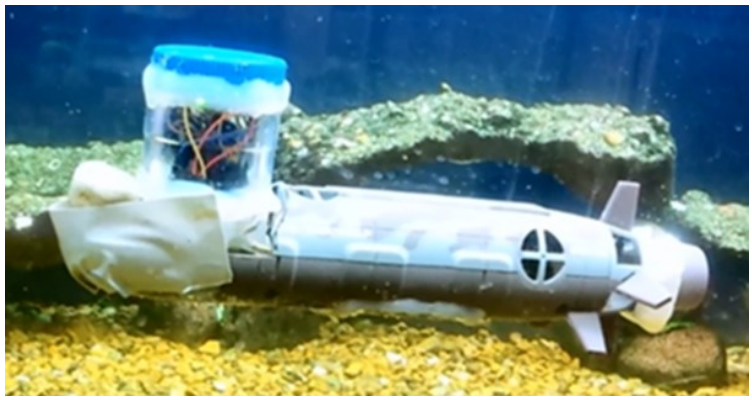

Fig. 13. Submarine depth tracking. The submarine is maintaining its depth in a large aquarium.

In order to see if the battery life specification was met, multiple tests were conducted. The first test conducted was an in-water, rear motor current draw test. The rear portion of the submarine was submerged and the average current draw of the motor was recorded to be 850 mA. The next test conducted was a bench-top average current draw test of the entire system. All of the hardware was connected as it would be underwater and software was run to simulate the current draw that would occur during underwater navigation. Because of the two power supplies coming off the batteries, two multimeters were required to measure the total current draw. The total current draw for this test was measured to be 1202 mA, as seen in the Appendix E. This bench-top test produced the same current draw that would be seen during an underwater test, with the exception of the motor currents. In the bench-top test, two motors were being powered in air, drawing 450 mA. This total motor current draw could be much higher underwater, especially if all three motors were being run at full duty cycle. The total motor current draw in this worst-case scenario is shown in equation 1. To calculate the worst-case battery life, the bench-top motor current is replaced with the worst-case underwater motor currents. The total current draw is shown in equation 2. The resulting worst-case battery life is shown in equation 3. The battery life in this worst-case scenario meets the 25 minute specification.

$$850\ mA * 3 = 2550\ mA \quad (1)$$

$$1202\ mA + 2550\ mA - 450\ mA = 3302\ mA \quad (2)$$

$$\frac{2500\ mAh}{3302\ mA} * 60\ minutes = 45\ minutes \quad (3)$$

The entire system was tested in a small swimming pool, as shown in Fig. 14. The only hardware that was not on the submarine during this testing was the camera system, as it was being tested separately. Videos were taken that demonstrate the mobility and autonomy of one of the submarines underwater (see Appendix G). In the mobility testing video, the forward navigation ability and turn radius of a submarine is shown. Despite adding a relatively large sail to the submarine, it is still mobile underwater. The calibration circle test shown in that video shows the submarine doing a circle in order to read ambient light values. The readings are then used to calibrate the photodiodes to ignore the ambient light. The autonomous nature of the submarines can also be seen in the mobility testing video. The video demonstrates the autonomous nature by showing that the submarine is completely autonomous once it has been placed in water. The submarine begins its navigation by doing the calibration startup circle. It will then begin navigating until it measures the battery voltage dropping below 10%. The submarine will then surface and move to the end of the testing pool where it can be easily retrieved.

Fig. 14. Submarine navigating underwater. The mobility of the platform was tested in a pool.

IV. Conclusions

The AUR team was able to get two individual submarines fully constructed. Setbacks prevented the group from performing testing with the entire swarm in water. These setbacks primarily involved the hardware and the waterproofing methods. The biggest setback was the design of the detection array. Despite this, the team was able to get all the subsystems functioning individually. In the case of the depth tracking, three of the four subsystems were integrated successfully. The camera system was completed and ready for underwater operation. The result was a submarine that could submerge, maintain its depth, and maneuver underwater autonomously.

V. Acknowledgements

The authors would like to thank Dr. Joseph Driscoll for his invaluable support and expertise throughout the duration of the project.

VI. References

[1] NOAA (2014). How much of the ocean have we explored? [Online]. Available: http://oceanservice.noaa.gov/facts/exploration.html

[2] NOAA (2014). What does an oceanographer do? [Online]. Available: http://oceanservice.noaa.gov/facts/oceanographer.html

[3] M. Patterson A. Trembanis (2014). AUVs: marine science's newest tool [Online]. Available: http://oceanexplorer.noaa.gov/explorations/08bonaire/background/auvs/auvs.html

[4] J. Yuh. "Design and control of autonomous underwater robots: a survey" Autonomous Systems Laboratory, University of Hawaii, Honolulu, Hawaii, 2000.

[5] J. H. Roach, B. B. Thompson, and R. J. Marks II. "Mapping an underwater minefield with a multi-state swarm and the effects of swarm size on performance" unpublished.

[6] T. Schmickl, R. Thenius, C. Moeslinger, J. Timmis, A. Tyrrell, M. Read, J. Hilder, J. Halloy, A. Campo, C. Stefanini, L. Manfredi, S. Orofino, S. Kernbach, T. Dipper, D. Sutantyo. "CoCoRo - the self-aware underwater swarm" unpublished.

[7] H. Bohm and V. Jensen, Build Your Own Underwater Robot and Other Wet Projects, 11th ed. Vancouver, B. C. Canada: Westcoast Words, 2012.

[8] H. Brundage. “Designing a Wireless Underwater Optical Communication System” unpublished.

[9] (2014). NIMH discharge curve [Online]. Available: http://lipomanufacturer.blogspot.com/2014_03_01_archive.html

[10] S. Sugihara (2014). Three rules of swarm algorithm [Online]. Available: http://igeo.jp/tutorial/43.html

VI. Appendix

(A) Parts List

TABLE V
PARTS LIST

Quantity	Description
10	20 Megaohms Resistor
3	.22 Ohms Resistor
2	1.8k Ohms Resistor
3	100 Ohms Resistor
3	1 Ohm Resistor
6	Transistor
1	IMU
1	4 pack AA Rechargeable Batteries
3	I2C H-bridge
2	Female-Female Wire
1	Camera Module
1	Pressure Sensor
1	Motorworks Submarine
1	Plastic Cylinder and Box
3	Blue LED
1	5V Step-Up Voltage Regulator
1	Adafruit Pro Trinket
1	Black Electrical Tape
1	Perf Board
1	Multiplexer
2	Op-amp
5	Photodiode
1	H-Bridge PCB
1	LED Driver PCB
1	Photodiode PCB
1	Solder Paste
1	Hot Glue
1	Diode
1	Capacitor

(B) Glossary

ADC - a circuit that converts analog signals into a stream of digital data.
Brownout - A condition where the voltage supplied to the system falls below the specified operating range, but above 0V.
Current - Ampere(s), the unit of electrical current. Current is defined as the amount of charge that flows past a given point, per unit of time.
Diode - A two-terminal device that rectifies signals (passes current in only one direction). Most commonly, a semiconductor consisting of a P-N junction, but diodes can also be realized using vacuum tube, point-contact, metal-semiconductor junction (Schottky), and other technologies.
H-bridge - A circuit diagram which resembles the letter "H." The load is the horizontal line, connected between two pairs of intersecting lines. It is very common in DC motor-drive applications where switches are used in the "vertical" branches of the "H" to control the direction of current flow, and thus the rotational direction of the motor.
I²C - I²C is a two-wire, low-speed, serial data connection IC bus used to run signals between integrated circuits, generally on the same board.
I/O - Input/output
Light-emitting diode - A semiconductor device that emits light (usually visible or infrared) when forward-biased.
Multiplexer - Combining two signals (which can be analog or a digital stream) into one in such a way that they can later be separated.
Op-amp - The ideal op amp is an amplifier with infinite input impedance, infinite open-loop gain, zero output impedance, infinite bandwidth, and zero noise. It has positive and negative inputs which allow circuits that use feedback to achieve a wide range of functions.
PCB - A printed circuit board is a non-conductive material with conductive lines printed or etched. Electronic components are mounted on the board and the traces connect the components together to form a working circuit or assembly.
PWM - A method for using pulse width to encode or modulate a signal. The width of each pulse is a function of the amplitude of the signal.
Resistance - Resistance, represented by the symbol R and measured in ohms, is a measure of the opposition to electrical flow in DC systems.
Switching regulator - A voltage regulator that uses a switching element to transform the supply into an alternating current, which is then converted to a different voltage using capacitors, inductors, and other elements, then converted back to DC.
Voltage - Unit of measure for electromotive force (EMF), the electrical potential between two points.

Glossary definitons have been aquired from:
http://www.maximintegrated.com/en/glossary/definitions.mvp/terms/index

(C) Detailed cost analysis

TABLE VI
DETAILED BILL OF MATERIALS

Quantity	Description	Price	Extended Price
10	20 Megaohms Resistor	$0.22	$2.22
3	.22 Ohms Resistor	$0.45	$1.34
2	1.8k Ohms Resistor	$0.08	$0.16
3	100 Ohms Resistor	$0.19	$0.58
3	1 Ohm Resistor	$0.47	$1.42
6	Transistor	$0.52	$3.14
1	IMU	$9.95	$9.95
1	4 pack AA Rechargeable Batteries	$5.99	$5.99
3	I2C H-bridge	$2.44	$7.32
2	Female-Female Wire	$2.00	$4.00
1	Camera Module	$11.99	$11.99
1	Pressure Sensor	$16.09	$16.09
1	Motorworks Submarine	$61.99	$61.99
1	Plastic Cylinder and Box	$2.60	$2.60
3	Blue LED	$2.62	$7.86
1	5V Step-Up Voltage Regulator	$3.95	$3.95
1	Adafruit Pro Trinket	$9.95	$9.95
1	Black Electrical Tape	$1.24	$1.24
1	Perf Board	$2.25	$2.25
1	Multiplexer	$0.44	$0.44
2	Op-amp	$3.60	$7.20
5	Photodiode	$0.38	$1.9
1	H-Bridge PCB	$2.68	$2.68
1	LED Driver PCB	$1.86	$1.86
1	Photodiode PCB	$1.58	$1.58
1	Solder Paste	$3.00	$3.00
1	Hot Glue	$0.5	$0.50
1	Diode	$0.3	$0.30
1	Capacitor	$0.2	$0.20
Total per submarine			$173.71
Total for swarm			$694.83

(D) Detailed schematics, models, flowcharts

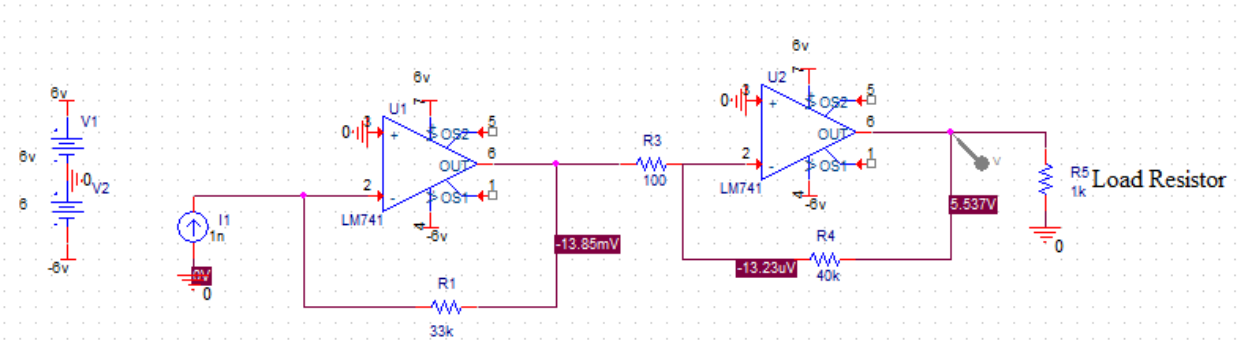

Fig. 15. First Op-Amp design for photodiode.

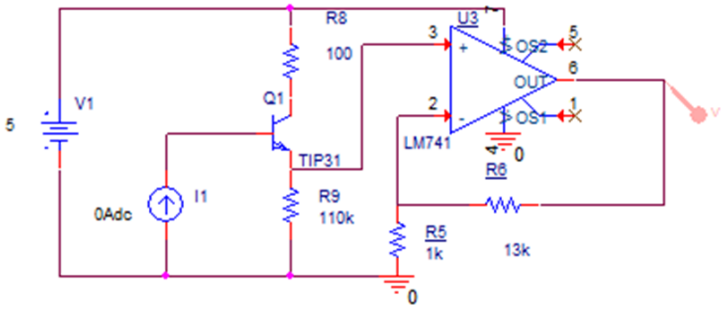

Fig. 16. Second Op-Amp design for photodiode using transistor.

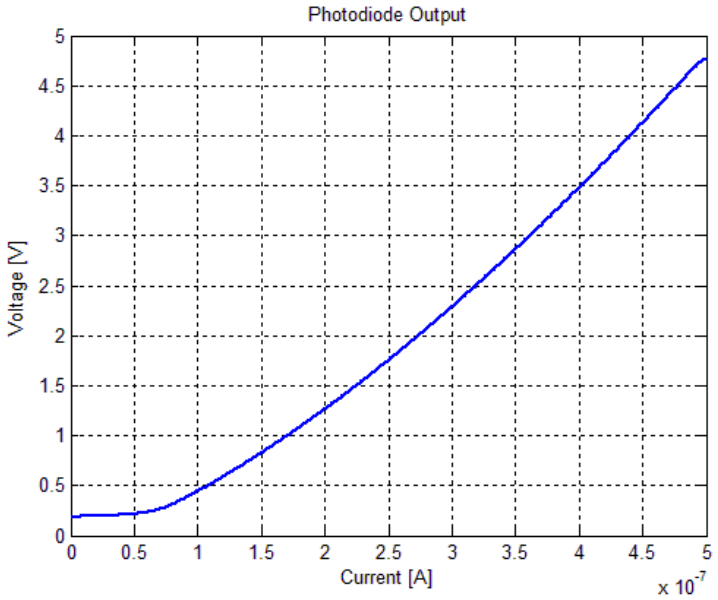

Fig. 17. Second Op-Amp design for photodiode using transistor output.

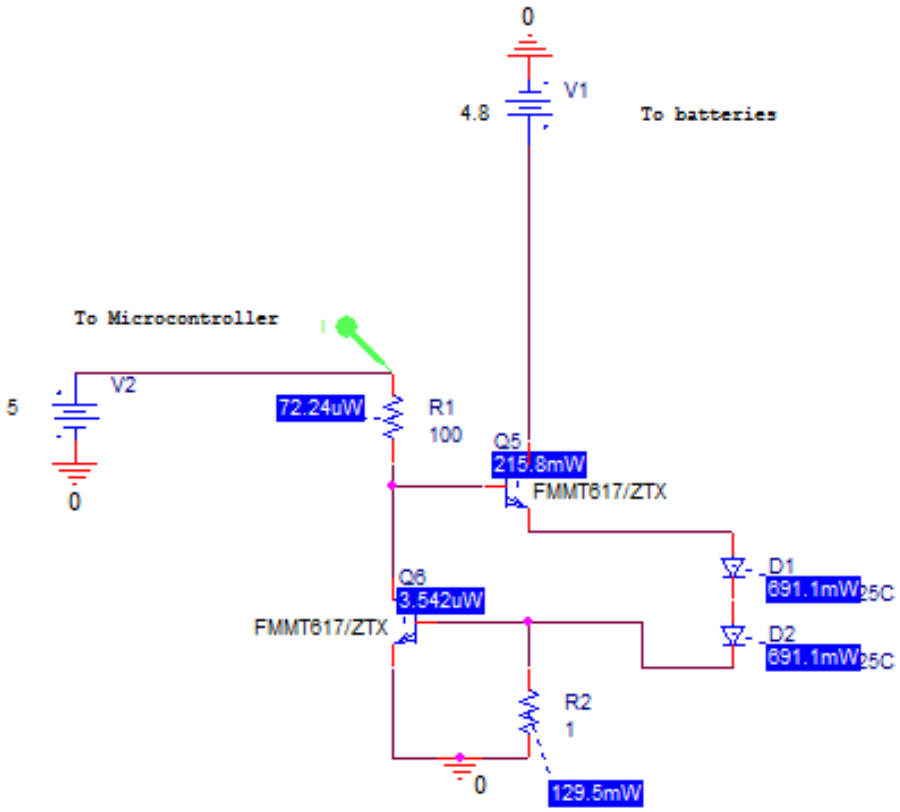

Fig. 18. Current source designed in OrCAD for 3W LED.

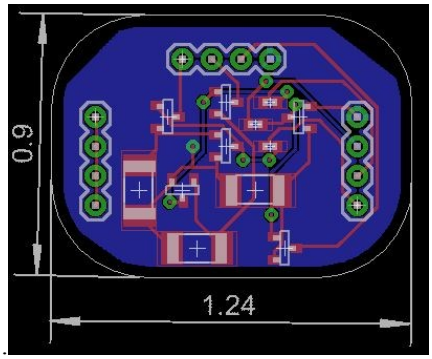

Fig. 19. LED Driver PCB schematic.

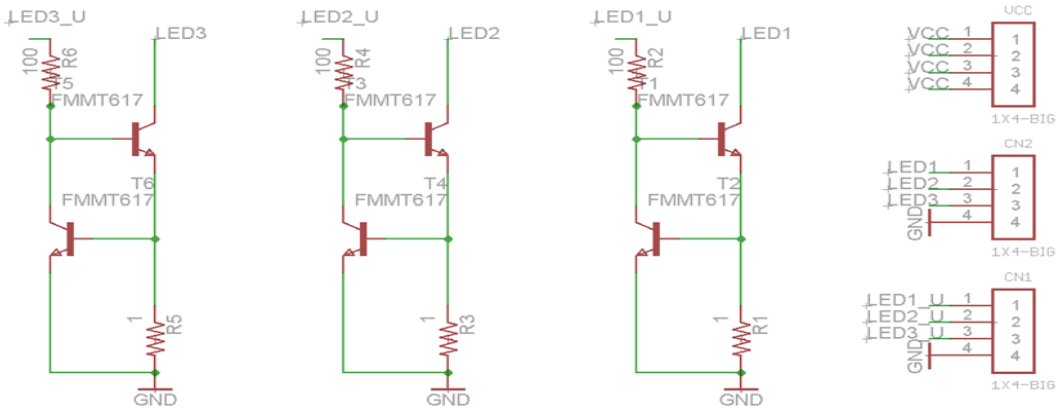

Fig. 20. LED Driver schematic.

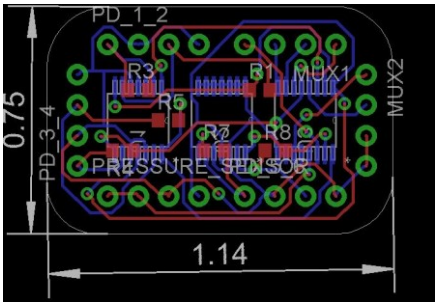

Fig. 21. Photodiode PCB schematic.

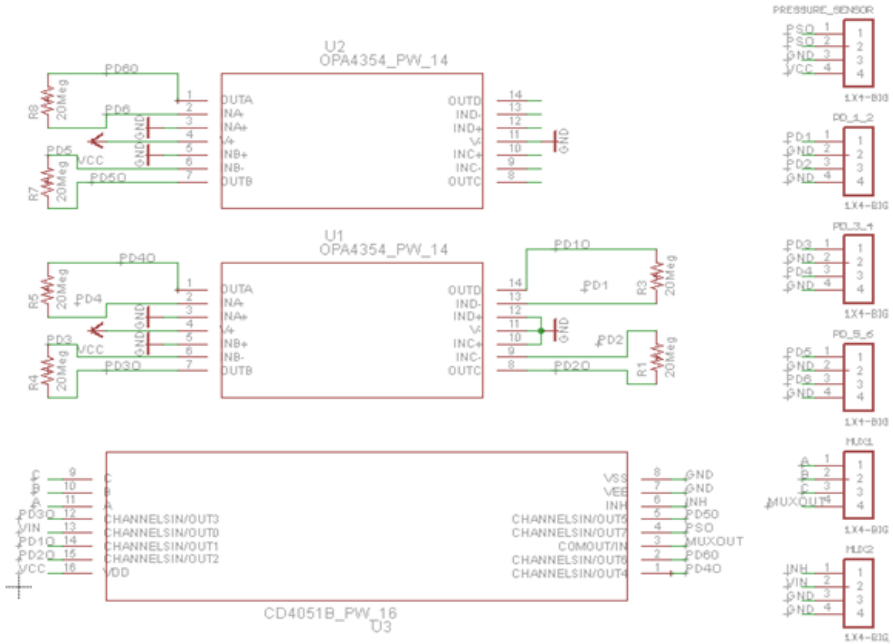

Fig. 22. Photodiode schematic.

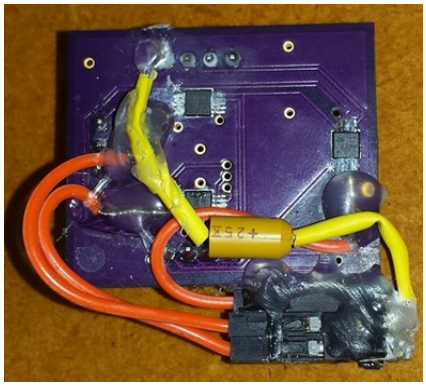

Fig. 23. Motor driver PCB.

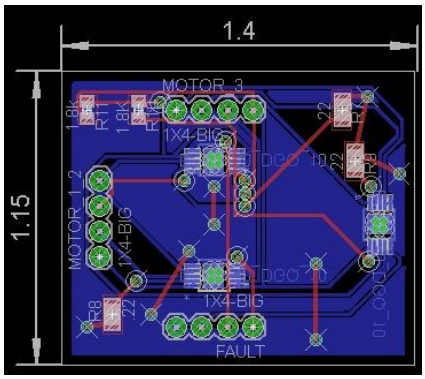

Fig. 24. Motor driver PCB schematic.

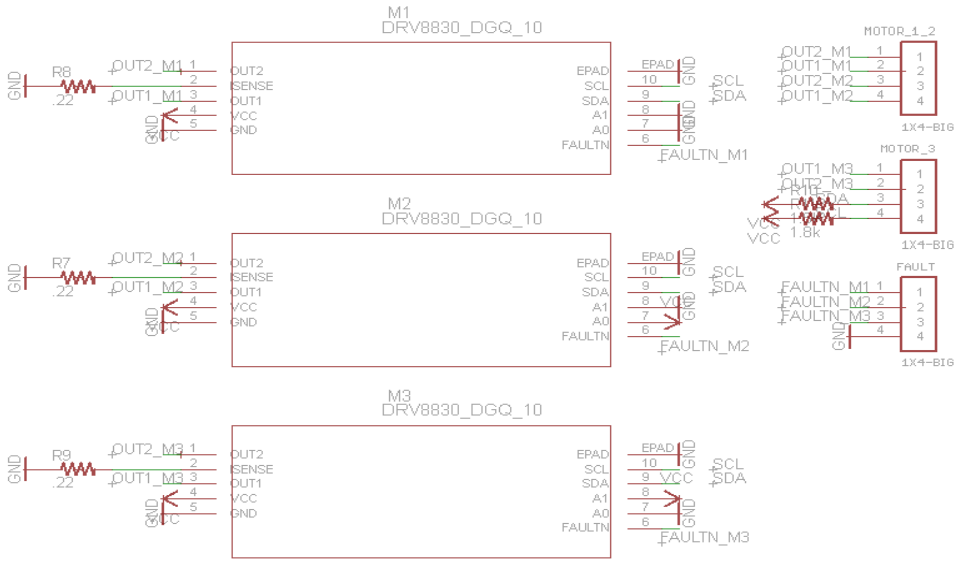

Fig. 25. Motor driver schematic.

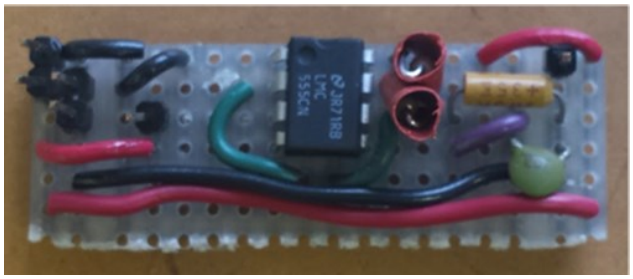

Fig. 26. 555 timer circuit. The circuit was soldered to a perf board.

Fig. 27. Camera PCB. This circuit was extracted from a key fob.

(E) Detailed experimental results

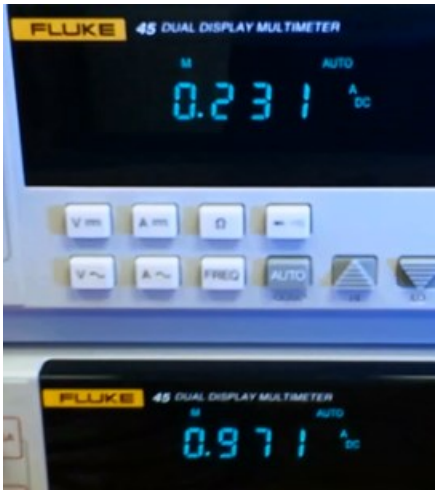

Fig. 28. The bench-top average current test readings. The 3 cell NIMH supply is on top.

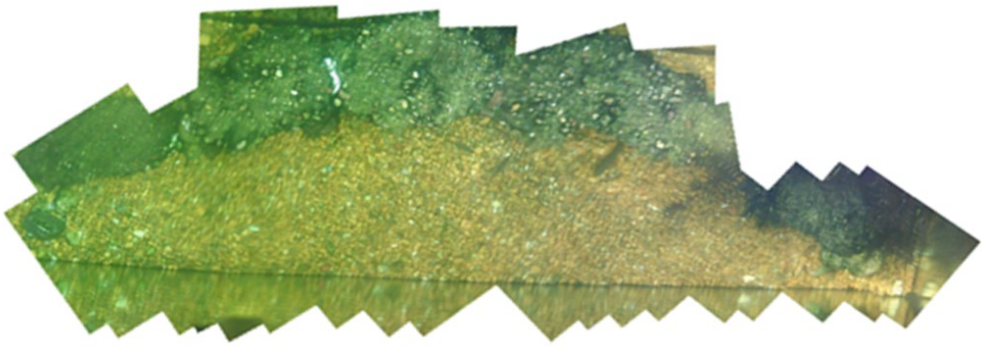

Fig. 29. Photo-merging result of an aquarium.

(f) Theory of operation

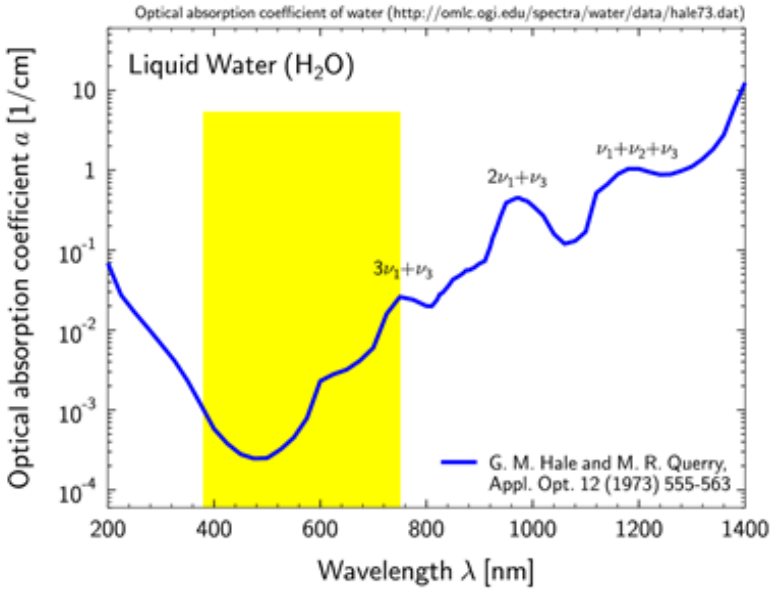

Fig. 30. The total optical absorption coefficient of water [8]. Figure shows the attenuation of light with various wavelengths in water. Note that the visible spectrum is seen highlighted in yellow.

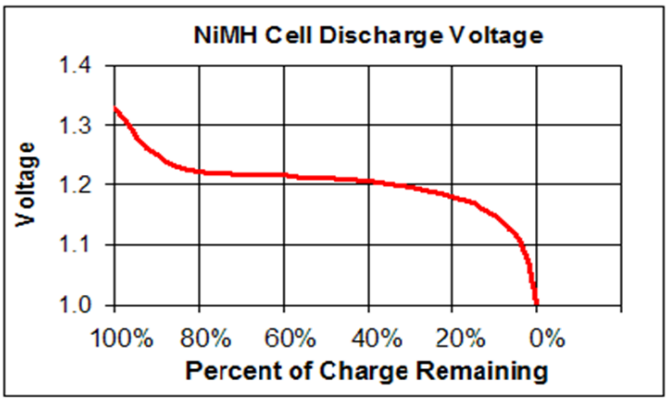

Fig. 31. The discharge voltage curve for a NIMH 1 cell battery [9]. This data was used in battery life estimation.

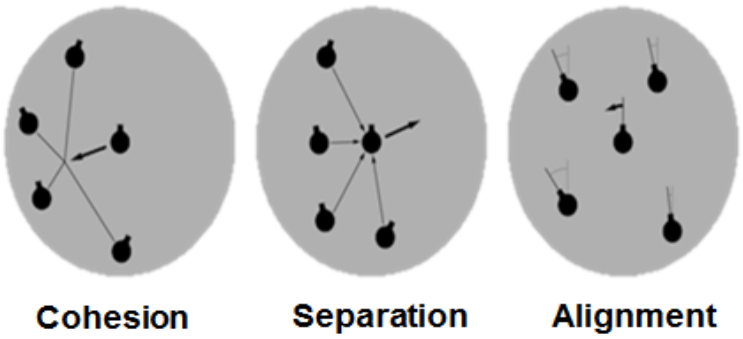

Fig. 32. Boids algorithm criteria [10]. This information is required for swarming.

(G) Video, web links

AUR Project Website: http://cegt201.bradley.edu/projects/proj2015/autonomous_underwater_robots/index.html
Submarine Mobility Testing Video: https://www.youtube.com/watch?v=wOSrQxpOtXE

Submarine Depth Tracking Video 1: https://www.youtube.com/watch?v=KP_CVridH08

Submarine Depth Tracking Video 2: https://www.youtube.com/watch?v=-tTprnzq--I

Submarine Depth Tracking Video 3: https://www.youtube.com/watch?v=9dOgWr1T1QA

(H) Code and Documentation

Main.c

```
/**
 * @file Main.c
 * @author Ryan Lipski, Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/11/2015\n Created: 11/11/2014 11:57:32 AM
 * @brief This file is code main for testing and the main program.
 */

#include "Main_Program.h"
#include "Master_Build_Control.h"
#include "Testing/Test_Battery.h"
#include "Testing/Test_H_Bridge.h"
#include "Testing/Test_I2C_Compass.h"
#include "Testing/Test_LED.h"
#include "Testing/Test_Motor_Control.h"
#include "Testing/Test_Multiplexer.h"
#include "Testing/Test_Photodiodes.h"
#include "Testing/Test_Pressure_Sensor.h"
#include "Testing/Test_Swarming.h"

/**
 * @brief This the main for the entire project use Main_Mode to select the
portion of the code to test.
 */
int main(void)
```

```
{
    /* Main for the entire project. */
    #if Main_Mode == 1
        Main_Program();
    /* Test the swarming code. */
    #elif Main_Mode == 2
        Test_Swarming();
    /* Test the LSM303 (I2C Compass/Accelerometer). */
    #elif Main_Mode == 3
        Test_I2C_Compass();
    /* Test the DRV8830 (I2C H-Bridge). */
    #elif Main_Mode == 4
        Test_H_Bridge();
    /* Test the LED toggling. */
    #elif Main_Mode == 5
        Test_LED();
    /* Test the Multiplexer and the ADC. */
    #elif Main_Mode == 6
        Test_Multiplexer();
    /* Test the photodiode circuits. */
    #elif Main_Mode == 7
        Test_Photodiodes();
    /* Test the pressure sensor. */
    #elif Main_Mode == 8
        Test_Pressure_Sensor();
    /* Test the battery reading code. */
    #elif Main_Mode == 9
        Test_Battery();
    /* Test the motor control. */
    #elif Main_Mode == 10
        Test_Motor_Control();
    /* Test the swarming code. */
    #elif Main_Mode == 11
        Test_Swarming();
    #endif
    /* Return of 0 for main. */
    return(0);
}
```

Master_Build_Control.h

```
/**
 * @file Master_Build_Control.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 3/20/2015\n Created: 2/26/2015 4:23:20 PM
 * @brief This is the header file to control which program builds.
 */

#ifndef MASTER_BUILD_CONTROL_H_
#define MASTER_BUILD_CONTROL_H_

/**
 * @brief Different modes for the main function\n
 * Mode 1: Main for the entire project.\n
 * Mode 2: Test the swarming code.\n
 * Mode 3: Test the LSM303 (I2C Compass/Accelerometer).\n
```

```
 * Mode 4: Test the DRV8830 (I2C H-Bridge).\n
 * Mode 5: Test the LED toggling.\n
 * Mode 6: Test the Multiplexer and the ADC.\n
 * Mode 7: Test the photodiode circuits.\n
 * Mode 8: Test the pressure sensor.\n
 * Mode 9: Test the battery reading code.\n
 * Mode 10: Test the motor control.\n
 * Mode 11: Test the swarming code.\n
 */
#define Main_Mode 1
/**
 * @brief Different modes for the compass function\n
 * Compass Test Mode 0: Test Acceleration and Magnetometer Readings\n
 * Compass Test Mode 1: Test Heading\n
 */
#define Compass_Test_Mode 0
/**
 * @brief Different modes for the h-bridge test function\n
 * H-Bridge Test Mode 0: Test X-axis\n
 * H-Bridge Test Mode 1: Test Y-axis\n
 * H-Bridge Test Mode 2: Test Z-axis\n
 */
#define H_Bridge_Test_Mode 0
/**
 * @brief Different modes for the h-bridge test function\n
 * H-Bridge Test Direction 0: Test goes from zero to max then repeats\n
 * H-Bridge Test Direction 1: Test goes from max to zero then repeats\n
 */
#define H_Bridge_Test_Direction 0
/**
 * @brief Different modes for the LED function\n
 * LED Test Mode 0: Test the toggling of the front LED\n
 * LED Test Mode 1: Test changing between 3 LEDs\n
 */
#define LED_Test_Mode 0
/**
 * @brief Different modes for the multiplexer function\n
 * Multiplexer Test Mode 0: Test a single pin\n
 * Multiplexer Test Mode 1: Test all of the pins\n
 * Multiplexer Test Mode 2: Test a single voltage conversion\n
 * Multiplexer Test Mode 3: Test voltage conversion on all of the pins\n
 */
#define Multiplexer_Test_Mode 0
/**
 * @brief Different modes for the photodiode function\n
 * Photodiode Test Mode 0: Test front diode\n
 * Photodiode Test Mode 1: Test left side diodes\n
 * Photodiode Test Mode 2: Test right side diodes\n
 * Photodiode Test Mode 3: Test rear diode\n
 * Photodiode Test Mode 4: Test all diodes\n
 * Photodiode Test Mode 5: Test all diodes and change the LEDs\n
 */
#define Photodiode_Test_Mode 0
/**
 * @brief Different modes for the motor control test function.\n
 * Motor Control Test Mode 0: Varying Test Forward/Reverse\n
 * Motor Control Test Mode 1: Varying Test Left/Right\n
```

```
 * Motor Control Test Mode 2: Varying Test Up/Down\n
 * Motor Control Test Mode 0: Constant Test Forward/Reverse\n
 * Motor Control Test Mode 1: Constant Test Left/Right\n
 * Motor Control Test Mode 2: Constant Test Up/Down\n
 */
#define Motor_Control_Test_Mode 0
/**
 * @brief Prints the different regions for the swarming code. 1: On / 0: Off
 */
#define Print_Region 0
/**
 * @brief Prints the information for motor control code. 1: On / 0: Off
 */
#define Print_Motor_Information 0
/**
 * @brief Prints the different state information for the state machine code.
1: On / 0: Off
 */
#define Print_State 0
/**
 * @brief Prints the different information for the h bridge update code. 1
for on 0 for off.
 */
#define H_Bridge_Print 0
/**
 * @brief Selects which state to test after the initialization. Note that
this has to be > 0.
 */
#define State_Machine_Test_Mode 0
#endif /* MASTER_BUILD_CONTROL_H_ */
```

Main_Program.c

```
/**
 * @file Main_Program.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 3/20/2015\n Created: 2/28/2015 5:50:34 PM
 * @brief This file is code for the main program.
 */

#include "Algorithms/State_Machine/State_Machine.h"

/**
 * @brief This is the main program for the project.
 */
void Main_Program(void)
{
    while(1)
    {
        /* Check to see if the state machine needs to be updated. */
        State_Machine_Update();
    }
}
```

Main_Program.h

```
/**
 * @file Main_Program.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 3/20/2015\n Created: 2/28/2015 5:58:00 PM
 * @brief This is the header file for the main program.
 */

#ifndef MAIN_PROGRAM_H_
#define MAIN_PROGRAM_H_

void Main_Program(void);

#endif /* MAIN_PROGRAM_H_ */
```

Motor_Control.c

```
/**
 * @file Motor_Control.c
 * @author Ryan Lipski, Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/28/2015\n Created: 1/27/2015 4:45:33 PM
 * @brief This file is code for the control loops for the three motors.
 */

/** @brief The Z axis motor constant. */
#define motorZ_const 1
/** @brief The proportional gain Z. */
#define kpZ 2.0
/** @brief The integral gain Z. */
#define kiZ 2.0
/** @brief The derivative gain Z. */
#define kdZ 1.0
/** @brief Time difference for the velocity calculation. */
#define Velocity_Time_Difference 0.05
/** @brief Motor control type for the Z motor. */
#define Motor_Z_Control_Type 1

#include "../../Master_Build_Control.h"
#include "../../Subsystem/H_Bridge/DRV8830.h"
#include "../../Subsystem/IMU/LSM303.h"
#include "../../Subsystem/Pressure/MPX5010GP.h"
#include "Motor_Control.h"

#if Print_Motor_Information == 1
    #include <stdio.h>
    #include "../../Microcontroller/Communication/USART.h"
#endif

/* @brief Desired velocity of the x-axis motor. */
static volatile float v_desX = 0.0;
/* @brief Desired velocity of the x-axis motor. */
static volatile float v_desY = 0.0;
```

```
/* @brief Desired depth of the submarine. */
static volatile float p_desZ = 2.0;
static uint8_t Update_Control_Flag = 1;
static uint8_t Update_Motor_X_Flag = 0;
static uint8_t Update_Motor_Y_Flag = 0;
static uint8_t Update_Motor_Z_Flag = 0;

void Motor_Control_X_Open()
{
    DRV8830_Set_Voltage_Motor_X(v_desX/5.0*Motor_Max_Voltage);
}

void Motor_Control_Y_Open()
{
    DRV8830_Set_Voltage_Motor_Y(v_desY/5.0*Motor_Max_Voltage);
}

void Motor_Control_Z(float depth)
{
    float errZ = p_desZ - depth; //calculate the error in position
    #if Motor_Z_Control_Type == 0
        if(errZ>0.1)
        {
            DRV8830_Set_Voltage_Motor_Z(Motor_Max_Voltage/2);
        }
        else if(errZ<-0.1)
        {
            DRV8830_Set_Voltage_Motor_Z(-Motor_Max_Voltage);
        }
        else
        {
            DRV8830_Set_Voltage_Motor_Z(-Motor_Max_Voltage/2);
        }
    #elif Motor_Z_Control_Type == 1
        if(errZ<0.1 && errZ>-0.1)//dead band for depth
        {
            errZ=0;
        }
        static float errZ_old = 0;//old error signal for motor 1
        static float errZ_old_2 = 0;//old error signal for motor
        float rZ = kpZ*(errZ-errZ_old_2)+kiZ*(errZ+errZ_old_2)/2+kdZ*(errZ-
2*errZ_old+errZ_old_2);
        errZ_old_2 = errZ_old;
        errZ_old = errZ;
        DRV8830_Set_Voltage_Motor_Z(rZ*motorZ_const);//set the motor 3's
voltage using 5.14 * (VSET)/64 = Vout
        #if Print_Motor_Information == 1
            char Buffer[100] = {0x00};
            char Buffer_Depth[FLOATING_POINT_BUFFER_SIZE] = {0x00};
            char Buffer_Error[FLOATING_POINT_BUFFER_SIZE] = {0x00};
            char Buffer_Voltage[FLOATING_POINT_BUFFER_SIZE] = {0x00};
            sprintf(Buffer,"Depth: %s errZ %s zvoltage %s\r\n",
USART_ftoa(depth, Buffer_Depth), USART_ftoa(errZ, Buffer_Error),
USART_ftoa(rZ*motorZ_const, Buffer_Voltage));
            USART_Send_String(Buffer);
        #endif
    #endif
```

```
}

/**
 * @brief This function is the control loops for the three motors.
 */
void Motor_Control(void)
{
    if(Update_Motor_X_Flag)
    {
        Motor_Control_X_Open();
    }
    if(Update_Motor_Y_Flag)
    {
        Motor_Control_Y_Open();
    }
    if(Update_Motor_Z_Flag)
    {
        Motor_Control_Z(MPX5010GP_Depth());
    }
}

/**
 * @brief This function is the desired velocity for the X-axis motor.
 * @param Updated_Value The new desired velocity for the X-axis motor.
 */
void Motor_Control_Update_Velocity_X(float Updated_Value)
{
    v_desX = Updated_Value;
}

/**
 * @brief This function is the desired velocity for the Y-axis motor.
 * @param Updated_Value The new desired velocity for the Y-axis motor.
 */
void Motor_Control_Update_Velocity_Y(float Updated_Value)
{
    v_desY = Updated_Value;
}

/**
 * @brief @brief This function is the desired depth of the submarine.
 * @param Updated_Value The new desired depth for the submarine.
 */
void Motor_Control_Update_Depth_Z(float Updated_Value)
{
    p_desZ = Updated_Value;
}

/**
 * @brief This function controls the value for the update control motor x
flag.
 * @param Input Value to set the update control motor x flag.
 */
void Motor_Control_Motor_X_Flag_Update(uint8_t Input)
{
    Update_Motor_X_Flag = Input;
}
```

```
/**
 * @brief This function controls the value for the update control motor y
flag.
 * @param Input Value to set the update control motor y flag.
 */
void Motor_Control_Motor_Y_Flag_Update(uint8_t Input)
{
    Update_Motor_Y_Flag = Input;
}

/**
 * @brief This function controls the value for the update control motor z
flag.
 * @param Input Value to set the update control motor z flag.
 */
void Motor_Control_Motor_Z_Flag_Update(uint8_t Input)
{
    Update_Motor_Z_Flag = Input;
}

/**
 * @brief This function is used to update the motor control if the update
flag is enabled.
 */
void Motor_Control_Update(void)
{
    if(Update_Control_Flag)
    {
        Motor_Control();
        Update_Control_Flag = 0;
    }
}

/**
 * @brief This function is used to enable the control flag.
 */
void Motor_Control_Enable_Flag(void)
{
    Update_Control_Flag = 1;
}
```

Motor_Control.h

```
/**
 * @file Motor_Control.h
 * @author Ryan Lipski, Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/28/2015\n Created: 1/27/2015 4:45:50 PM
 * @brief This file is code for the control loops for the three motors.
 */

#ifndef MOTOR_CONTROL_H_
#define MOTOR_CONTROL_H_

#include <stdint.h>
```

```
void Motor_Control(void);
void Motor_Control_Update_Velocity_X(float UpdateValue);
void Motor_Control_Update_Velocity_Y(float Updated_Value);
void Motor_Control_Update_Depth_Z(float UpdateValue);
void Motor_Control_Motor_X_Flag_Update(uint8_t Input);
void Motor_Control_Motor_Y_Flag_Update(uint8_t Input);
void Motor_Control_Motor_Z_Flag_Update(uint8_t Input);
void Motor_Control_Update(void);
void Motor_Control_Enable_Flag(void);

#endif /* MOTOR_CONTROL_H_ */
```

Swarming.c

```
/**
 * @file Swarming.c
 * @author Ryan Lipski, Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/12/2015 11:35:12 PM
 * @brief This file is code for the swarming algorithm.
 */

/** @brief The maximum ratio value of photodiodes to be considered "aligned."
*/
#define Photodiode_Ratio_Maximum 1.2
/** @brief The minimum ratio value of photodiodes to be considered "aligned."
*/
#define Photodiode_Ratio_Minimum 0.8

/** @brief The value associated with the various regions. */
enum Region_Value
{
    Red_Region_Value,
    Green_Region_Value,
    Yellow_Region_Value,
    Outside_Region_Value,
    Front_Region_Value,
    Center_Region_Value,
    Rear_Region_Value,
};

#include <stdint.h>
#include "../../Master_Build_Control.h"
#include "../../Subsystem/Light/CLS15.h"
#include "../../Algorithms/Motor_Control/Motor_Control.h"
#include "Swarming.h"

#if Print_Region == 1
    #include <stdio.h>
    #include "../../Microcontroller/Communication/USART.h"
#endif

/* X offset from the swarming code. */
static int X_Offset = 0;
```

```
/* Y offset from the swarming code. */
static int Y_Offset = 0;
/* Flag to signify that the swarming algorithm is to be updated. */
static int Update_Swarming_Flag = 1;

inline uint8_t Swarming_Region(int Current_Distance)
{
    if(Current_Distance>=Radius_Red)
    {
        return(Red_Region_Value);
    }
    else if(Current_Distance>=Radius_Green)
    {
        return(Green_Region_Value);
    }
    else if(Current_Distance>=Radius_Yellow)
    {
        return(Yellow_Region_Value);
    }
    else
    {
        return(Outside_Region_Value);
    }
}

inline uint8_t Swarming_Range_Left_Right(float Ratio)
{
    if(Ratio>Photodiode_Ratio_Maximum)
    {
        return(Front_Region_Value);
    }
    else if(Ratio<Photodiode_Ratio_Minimum)
    {
        return(Rear_Region_Value);
    }
    else
    {
        return(Center_Region_Value);
    }
}

inline void Swarming_Right_Calculation(int *X, int *Y, int
Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio)
{
    uint8_t Ratio_Region = Swarming_Range_Left_Right(Ratio);
    uint8_t Front_Region = Swarming_Region(Photodiode_Value_Front);
    uint8_t Rear_Region = Swarming_Region(Photodiode_Value_Rear);
    uint8_t Check_Region = Outside_Region_Value;
    #if Print_Region == 1
        char Buffer[20] = {0x00};
        sprintf(Buffer, "Left %d, %d", Photodiode_Value_Front,
Photodiode_Value_Rear);
        USART_Send_String(Buffer);
    #endif
    if(Front_Region <= Rear_Region)
    {
        Check_Region = Front_Region;
```

```
    }
    else
    {
        Check_Region = Rear_Region;
    }
    switch(Check_Region)
    {
        case Red_Region_Value: //check the near active region
            switch(Ratio_Region)
            {
                case Front_Region_Value:
                    //turn left b*Y a percentage and decrease forward speed
b*Y a percentage
                    *X+=2;//left = +, right = -
                    #if Print_Region == 1
                        USART_Send_String("Right front red region\n\r");
                    #endif
                    break;
                case Center_Region_Value:
                    //turn left b*Y a percentage
                    *X+=1;//left = +, right = -
                    #if Print_Region == 1
                        USART_Send_String("Right center red region\n\r");
                    #endif
                    break;
                case Rear_Region_Value:
                    //turn left b*Y a percentage and increase forward speed
b*Y a percentage
                    *X+=1;//left = +, right = -
                    *Y-=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Right rear red region\n\r");
                    #endif
                    break;
            }
            break;
        case Green_Region_Value:
            switch(Ratio_Region)
            {
                case Front_Region_Value:
                    //increase forward speed b*Y a percentage
                    *Y+=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Right front green region\n\r");
                    #endif
                    break;
                case Center_Region_Value:
                    #if Print_Region == 1
                        USART_Send_String("Right center green region\n\r");
                    #endif
                    break;
                case Rear_Region_Value:
                    //decrease forward speed b*Y a percentage
                    *Y-=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Right rear green region\n\r");
                    #endif
```

```
                    break;
            }
            break;
        case Yellow_Region_Value:
            switch(Ratio_Region)
            {
                case Front_Region_Value:
                    //turn right b*Y a percentage and decrease forward speed
b*Y a percentage
                    *X+=1;//left = +, right = -
                    *Y+=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Right front yellow region\n\r");
                    #endif
                    break;
                case Center_Region_Value:
                    //turn right b*Y a percentage
                    *X+=1;//left = +, right = -
                    #if Print_Region == 1
                        USART_Send_String("Right center yellow region\n\r");
                    #endif
                    break;
                case Rear_Region_Value:
                    //turn right b*Y a percentage and increase forward speed
b*Y a percentage
                    *X+=1;//left = +, right = -
                    *Y-=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Right rear yellow region\n\r");
                    #endif
                    break;
            }
            break;
        default:
            #if Print_Region == 1
                USART_Send_String("Right outside region\n\r");
            #endif
            break;
    }
}

inline void Swarming_Left_Calculation(int *X, int *Y, int
Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio)
{
    uint8_t Ratio_Region = Swarming_Range_Left_Right(Ratio);
    uint8_t Front_Region = Swarming_Region(Photodiode_Value_Front);
    uint8_t Rear_Region = Swarming_Region(Photodiode_Value_Rear);
    uint8_t Check_Region = Outside_Region_Value;
    #if Print_Region == 1
        char Buffer[20] = {0x00};
        sprintf(Buffer, "Left %d, %d", Photodiode_Value_Front,
Photodiode_Value_Rear);
        USART_Send_String(Buffer);
    #endif
    if(Front_Region <= Rear_Region)
    {
```

```
        Check_Region = Front_Region;
    }
    else
    {
        Check_Region = Rear_Region;
    }
    switch(Check_Region)
    {
        case Red_Region_Value: //check the near active region
            switch(Ratio_Region)
            {
                case Front_Region_Value:
                    //turn left b*Y a percentage and decrease forward speed
b*Y a percentage
                    *X-=2;//left = +, right = -
                    *Y-=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Left front red region\n\r");
                    #endif
                    break;
                case Center_Region_Value:
                    //turn left b*Y a percentage
                    *X-=1;//left = +, right = -
                    #if Print_Region == 1
                        USART_Send_String("Left center red region\n\r");
                    #endif
                    break;
                case Rear_Region_Value:
                    //turn left b*Y a percentage and increase forward speed
b*Y a percentage
                    *X-=1;//left = +, right = -
                    *Y+=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Left rear red region\n\r");
                    #endif
                    break;
            }
            break;
        case Green_Region_Value:
            switch(Ratio_Region)
            {
                case Front_Region_Value:
                    //increase forward speed b*Y a percentage
                    *Y+=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Left front green region\n\r");
                    #endif
                    break;
                case Center_Region_Value:
                    #if Print_Region == 1
                        USART_Send_String("Left center green region\n\r");
                    #endif
                    break;
                case Rear_Region_Value:
                    //decrease forward speed b*Y a percentage
                    *Y-=1;//forward = +, reverse = -
                    #if Print_Region == 1
```

```
                        USART_Send_String("Left rear green region\n\r");
                    #endif
                    break;
            }
            break;
        case Yellow_Region_Value:
            switch(Ratio_Region)
            {
                case Front_Region_Value:
                    //turn right b*Y a percentage and decrease forward speed
b*Y a percentage
                    *X-=1;//left = +, right = -
                    *Y+=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Left front yellow region\n\r");
                    #endif
                    break;
                case Center_Region_Value:
                    //turn right b*Y a percentage
                    *X-=1;//left = +, right = -
                    #if Print_Region == 1
                        USART_Send_String("Left center yellow region\n\r");
                    #endif
                    break;
                case Rear_Region_Value:
                    //turn right b*Y a percentage and increase forward speed
b*Y a percentage
                    *X-=1;//left = +, right = -
                    *Y-=1;//forward = +, reverse = -
                    #if Print_Region == 1
                        USART_Send_String("Left rear yellow region\n\r");
                    #endif
                    break;
            }
            break;
        default:
            #if Print_Region == 1
                USART_Send_String("Left outside region\n\r");
            #endif
            break;
    }
}

inline void Swarming_Forward_Calculation(int *X, int *Y, int
Photodiode_Value)
{
    #if Print_Region == 1
        char Buffer[20] = {0x00};
        sprintf(Buffer, "Front %d", Photodiode_Value);
        USART_Send_String(Buffer);
    #endif
    switch(Swarming_Region(Photodiode_Value))
    {
        case Red_Region_Value: //check the near active region
            //turn left b*Y a percentage and reduce forward speed b*Y a
percentage
            *X=5;//left = +, right = -
```

```
            *Y=0;//forward = +, reverse = -
            #if Print_Region == 1
                USART_Send_String("Front red region\n\r");
            #endif
            break;
        case Green_Region_Value: //check the far active region
            //turn left b*Y a percentage and reduce forward speed b*Y a
percentage
            *X=5;//left = +, right = -
            *Y=0;//forward = +, reverse = -
            #if Print_Region == 1
                USART_Send_String("Front yellow region\n\r");
            #endif
            break;
        case Yellow_Region_Value: //check the far active region
            //turn left b*Y a percentage and reduce forward speed b*Y a
percentage
            *X=5;//left = +, right = -
            *Y=0;//forward = +, reverse = -
            #if Print_Region == 1
            USART_Send_String("Front yellow region\n\r");
            #endif
            break;
    }
}

int Swarming_Retrieve_X_Offset(void)
{
    return(X_Offset);
}

int Swarming_Retrieve_Y_Offset(void)
{
    return(Y_Offset);
}

void Swarming(void)
{
    X_Offset = 0; //*X-plane motor control variable, left=+ and right=-
movements
    Y_Offset = 3; //*Y-plane motor control variable, forward=+ and reverse=-
movements
    Swarming_Left_Calculation(&X_Offset, &Y_Offset,
CLS15_Retrieve_Photodiode_ADC_Left_Front(),
CLS15_Retrieve_Photodiode_ADC_Left_Rear(),
CLS15_Retrieve_Photodiode_Ratio_Left());
    Swarming_Right_Calculation(&X_Offset, &Y_Offset,
CLS15_Retrieve_Photodiode_ADC_Right_Front(),
CLS15_Retrieve_Photodiode_ADC_Right_Rear(),
CLS15_Retrieve_Photodiode_Ratio_Right());
    Swarming_Forward_Calculation(&X_Offset, &Y_Offset,
CLS15_Retrieve_Photodiode_ADC_Front());
}

void Swarming_Update(void)
{
    if(Update_Swarming_Flag)
```

```
    {
        Swarming();
        Motor_Control_Update_Velocity_X(Swarming_Retrieve_X_Offset());
        Motor_Control_Update_Velocity_Y(Swarming_Retrieve_Y_Offset());
        Motor_Control_Update_Depth_Z(2.0);
        Update_Swarming_Flag = 0;
    }
}

void Swarming_Enable_Flag(void)
{
    Update_Swarming_Flag = 1;
}
```

Swarming.h

```
/**
 * @file Swarming.h
 * @author Ryan Lipski, Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/12/2015 11:35:36 PM
 * @brief  This file is the header file for the swarming algorithm.
 */

#ifndef Swarming_H_
#define Swarming_H_

/** @brief Maximum radius for each of the ranges in inches. */
enum Region_Radius
{
    Radius_Red = 750,
    Radius_Green = 400,
    Radius_Yellow = 100
};

void Swarming(void);
int Swarming_Retrieve_X_Offset(void);
int Swarming_Retrieve_Y_Offset(void);
void Swarming_Update(void);
void Swarming_Enable_Flag(void);

#endif /* Swarming_H_ */
```

State_Machine.c

```
/**
 * @file State_Machine.c
 * @author Ryan Lipski, Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/12/2015 2:13:56 PM
 * @brief This file is code for the state machine code.
 */

#include "../../Algorithms/Motor_Control/Motor_Control.h"
#include "../../Algorithms/Navigation/Swarming.h"
#include "../../Master_Build_Control.h"
```

```
#include "../../Microcontroller/ADC/CD4051.h"
#include "../../Microcontroller/ADC/ADC.h"
#include "../../Microcontroller/Communication/I2C_Interface.h"
#include "../../Microcontroller/Interrupt/Timer.h"
#include "../../Subsystem/H_Bridge/DRV8830.h"
#include "../../Subsystem/Light/CLS15.h"
#include "../../Subsystem/Light/XPEBBL.h"
#include "../../Subsystem/IMU/LSM303.h"
#include "../../Subsystem/Pressure/MPX5010GP.h"
#include "../../Subsystem/Power/Battery.h"
#include "State_Machine.h"

#if Print_State == 1 || Print_Region == 1
    #include <stdio.h>
    #include "../../Microcontroller/Communication/USART.h"
#endif

static int Update_State_Flag = 1;

/** @brief Transition routine for between state 0 and 1. Initialize all
active subsystems. */
inline void State_Machine_Initialization(void)
{
    #if Print_State == 1 || Print_Region == 1
        USART_Initialize();
        USART_Send_String("State transition 1\n\r");
    #endif
    /* Initialize the TWI bus for the Compass and the H-bridge. */
    I2C_Interface_Initialize();
    /* Initialize the Compass subsystem. */
    //LSM303_Initialize();
    /* Initialize the h-bridges. */
    DRV8830_Initialize();
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize the LEDs. */
    XPEBBL_Initialize();
    /* Initialize the timers. */
    Timer_Initialize();
    /* Disable the motor control for the X motor. */
    Motor_Control_Motor_X_Flag_Update(0);
    /* Disable the motor control for the Y motor. */
    Motor_Control_Motor_Y_Flag_Update(0);
    /* Disable the motor control for the Z motor. */
    Motor_Control_Motor_Z_Flag_Update(0);
    /* Disable Timer 0. */
    Timer0_Disable();
    /* Disable Timer 1. */
    Timer1_Disable();
    /* Enable Timer 2 to update the state machine. */
    Timer2_Enable();
}

/** @brief Routine for state 1. Indicate that the submarine is waiting to be
depth. */
inline void State_Machine_State_1(void)
{
```

```
    #if Print_State == 1
        USART_Send_String("State 1\n\r");
    #endif
    /* Indicate by toggling the front LED. */
    XPEBBL_Toggle_Front_LED();
}

/** @brief Routine for state 2. Update the heading for going forward. */
inline void State_Machine_State_2(void)
{
    #if Print_State == 1
        USART_Send_String("State 2\n\r");
    #endif
    /* Update what is defined as the heading for going forward. */
    LSM303_UpdateForwardHeading();
    /* Update the motor control system. */
    Motor_Control_Update();
}

/** @brief Routine for state 3. Calibrate the offsets for the photodiodes. */
inline void State_Machine_State_3(void)
{
    #if Print_State == 1
        USART_Send_String("State 3\n\r");
    #endif
    /* Set the turning to maximum. */
    DRV8830_Set_Voltage_Motor_X(Motor_Max_Voltage);
    /* Read the photodiodes and update the offset if needed. */
    CLS15_Update_Photodiode_Offset();
    /* Update the motor control system. */
    Motor_Control_Update();
}

/** @brief Routine for state 4. Normal operation for the submarine. */
inline void State_Machine_State_4(void)
{
    #if Print_State == 1
        USART_Send_String("State 4\n\r");
    #endif
    /* Update the photodiodes taking into account the current active LED. */
    CLS15_Update_Photodiode_Values(XPEBLL_Change_LED());
    /* Update the swarming values using the current LED values. */
    Swarming_Update();
    /* Update the motor control system. */
    Motor_Control_Update();
}

/** @brief Routine for state 5. Toggling the front LED to indicate that . */
inline void State_Machine_State_5(void)
{
    #if Print_State == 1
        USART_Send_String("State 5\n\r");
    #endif
    /* Disable the turning motor. */
    DRV8830_Set_Voltage_Motor_X(0.0);
    /* Set the forward motor to maximum voltage. */
    DRV8830_Set_Voltage_Motor_Y(Motor_Max_Voltage);
```

```
    /* Update the front photodiodes taking into account the current active
LED. */
    CLS15_Update_Photodiode_Values(XPEBBL_Toggle_Front_LED()<<5);
    /* Update the motor control system. */
    Motor_Control_Update();
}

/** @brief Routine for state 6. Indicate that the submarine is done. */
inline void State_Machine_State_6(void)
{
    #if Print_State == 1
        USART_Send_String("State 6\n\r");
    #endif
    /* Disable the turning motor. */
    DRV8830_Set_Voltage_Motor_X(0.0);
    /* Disable the forward motor. */
    DRV8830_Set_Voltage_Motor_Y(0.0);
    /* Indicate by toggling the front LED. */
    XPEBBL_Toggle_Front_LED();
    /* Update the motor control system. */
    Motor_Control_Update();
}

inline void State_Machine_Trasition_2(void)
{
    #if Print_State == 1
        USART_Send_String("State transition 2\n\r");
    #endif
    /* Turn off the LEDs. */
    XPEBBL_Set_Pin(0x00);
    /* Enable the motor control for the X motor. */
    Motor_Control_Motor_X_Flag_Update(0);
    /* Enable the motor control for the Y motor. */
    Motor_Control_Motor_Y_Flag_Update(0);
    /* Enable the motor control for the Z motor. */
    Motor_Control_Motor_Z_Flag_Update(1);
    /* Enable Timer 0 to update the motor control. */
    Timer0_Enable();
    /* Disable Timer 1. */
    Timer1_Disable();
}

inline void State_Machine_Trasition_3(void)
{
    #if Print_State == 1
        USART_Send_String("State transition 3\n\r");
    #endif
    /* Set the turning to maximum. */
    DRV8830_Set_Voltage_Motor_X(Motor_Max_Voltage);
    /* Enable the motor control for the X motor. */
    Motor_Control_Motor_X_Flag_Update(0);
    /* Enable the motor control for the Y motor. */
    Motor_Control_Motor_Y_Flag_Update(0);
    /* Enable the motor control for the Z motor. */
    Motor_Control_Motor_Z_Flag_Update(1);
    /* Enable Timer 0 to update the motor control. */
    Timer0_Enable();
```

```
    /* Disable Timer 1. */
    Timer1_Disable();
}

inline void State_Machine_Trasition_4(void)
{
    #if Print_State == 1
        USART_Send_String("State transition 4\n\r");
    #endif
    /* Enable the motor control for the X motor. */
    Motor_Control_Motor_X_Flag_Update(1);
    /* Enable the motor control for the Y motor. */
    Motor_Control_Motor_Y_Flag_Update(1);
    /* Enable the motor control for the Z motor. */
    Motor_Control_Motor_Z_Flag_Update(1);
    /* Enable Timer 0 to update the motor control. */
    Timer0_Enable();
    /* Enable Timer 0 to update the swarm algorithm. */
    Timer1_Enable();
}

void State_Machine_Trasition_5(void)
{
    #if Print_State == 1
        USART_Send_String("State transition 5\n\r");
    #endif
    /* Disable the turning motor. */
    DRV8830_Set_Voltage_Motor_X(0.0);
    /* Set the forward motor to maximum voltage. */
    DRV8830_Set_Voltage_Motor_Y(Motor_Max_Voltage);
    /* Disable the motor control for the X motor. */
    Motor_Control_Motor_X_Flag_Update(0);
    /* Disable the motor control for the Y motor. */
    Motor_Control_Motor_Y_Flag_Update(0);
    /* Enable the motor control for the Z motor. */
    Motor_Control_Motor_Z_Flag_Update(1);
    /* Enable Timer 0 to update the motor control. */
    Timer0_Enable();
    /* Disable Timer 1. */
    Timer1_Disable();
}

void State_Machine_Trasition_6(void)
{
    #if Print_State == 1
        USART_Send_String("State transition 6\n\r");
    #endif
    /* Disable the turning motor. */
    DRV8830_Set_Voltage_Motor_X(0.0);
    /* Disable the forward motor. */
    DRV8830_Set_Voltage_Motor_Y(0.0);
    /* Set the motor control to surface. */
    Motor_Control_Update_Depth_Z(0);
    /* Disable the motor control for the X motor. */
    Motor_Control_Motor_X_Flag_Update(0);
    /* Disable the motor control for the Y motor. */
```

```
    Motor_Control_Motor_Y_Flag_Update(0);
    /* Enable the motor control for the Z motor. */
    Motor_Control_Motor_Z_Flag_Update(1);
    /* Enable Timer 0 to update the motor control. */
    Timer0_Enable();
    /* Disable Timer 1. */
    Timer1_Disable();
}

void State_Machine_Update(void)
{
    static int Current_State = 0;
    static int Previous_State = 0;
    if(Update_State_Flag)
    {
        switch(Current_State)
        {
            case 0:
                State_Machine_Initialization();
                #if State_Machine_Test_Mode > 0
                    Current_State = State_Machine_Test_Mode;
                #else
                    Current_State = 1;
                #endif
                break;
            case 1:
                State_Machine_State_1();
                #if Print_State == 1
                    char Buffer_State_1[30];
                    char Buffer_Depth[FLOATING_POINT_BUFFER_SIZE] = {0x00};
                    sprintf(Buffer_State_1, "%s
ft\n\r",USART_ftoa(MPX5010GP_Depth(), Buffer_Depth));
                    USART_Send_String(Buffer_State_1);
                #endif
                if(MPX5010GP_Depth() > 0.5)
                {
                    Current_State = 2;
                }
                break;
            case 2:
                State_Machine_State_2();
                uint8_t IsStabilized = LSM303_IsStablized();
                #if Print_State == 1
                    if(IsStabilized)
                    {
                        USART_Send_String("Is stabilized");
                    }
                    else
                    {
                        USART_Send_String("Is not stabilized");
                    }
                #endif
                if(IsStabilized)
                {
                    Current_State = 3;
                }
```

```
            break;
        case 3:
            State_Machine_State_3();
            #if Print_State == 1
                if(LSM303_RotationFinished())
                {
                    USART_Send_String("Rotation is finished\n");
                }
                else
                {
                    USART_Send_String("Rotation is not finished\n");
                }
            #endif
            if(LSM303_RotationFinished())
            {
                Current_State = 4;
            }
            break;
        case 4:
            State_Machine_State_4();
            uint8_t Motor_Status = DRV8830_Check_Motors();
            if(Motor_Status&0x02)
            {
                Current_State = 6;
            }
            else if(Battery_Check_Status()||(Motor_Status&0x05))
            {
                Current_State = 6;
            }
            break;
        case 5:
            State_Machine_State_5();
            if(CLS15_Retrieve_Photodiode_ADC_Front()<Radius_Red)
            {
                Current_State = 6;
            }
            break;
        case 6:
            State_Machine_State_6();
            break;
    }
    #if State_Machine_Test_Mode == 0
        if(Current_State != Previous_State)
        {
            switch(Current_State)
            {
                case 2:
                State_Machine_Trasition_2();
                break;
                case 3:
                State_Machine_Trasition_3();
                break;
                case 4:
                State_Machine_Trasition_4();
                break;
                case 5:
                State_Machine_Trasition_5();
```

```
                    break;
                    case 6:
                    State_Machine_Trasition_6();
                    break;
                }
            }
        #else
            if(Current_State != Previous_State)
            {
                switch(Current_State)
                {
                    case 2:
                    State_Machine_Trasition_2();
                    break;
                    case 3:
                    State_Machine_Trasition_3();
                    break;
                    case 4:
                    State_Machine_Trasition_4();
                    break;
                    case 5:
                    State_Machine_Trasition_5();
                    break;
                    case 6:
                    State_Machine_Trasition_6();
                    break;
                }
            }
            else
            {
                Current_State = Previous_State;
            }
        #endif
        Previous_State = Current_State;
        Update_State_Flag = 0;
        #if Print_State == 1
        char Buffer_State_4[100];
        char Buffer_State_4_2[FLOATING_POINT_BUFFER_SIZE];
        sprintf(Buffer_State_4, "Battery Status: %d, %s V\n\r",
Battery_Check_Status(),
USART_ftoa(ADC_Convert_To_Voltage(CLS15_Retrieve_Photodiode_ADC_Front()),Buff
er_State_4_2));
        USART_Send_String(Buffer_State_4);
        #endif
    }
}

void State_Machine_Enable_Flag(void)
{
    Update_State_Flag = 1;
}
```

State_Machine.h

```
/**
 * @file State_Machine.h
 * @author Nicholas Sikkema
```

```
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/12/2015 4:00:28 PM
 * @brief This file is the header file for the state machine code.
 */

#ifndef STATE_MACHINE_H_
#define STATE_MACHINE_H_

void State_Machine_Enable_Flag(void);
void State_Machine_Update(void);

#endif /* STATE_MACHINE_H_ */
```

ADC.c

```
/**
 * @file ADC.c
 * @author Nicholas Sikkema
 * @version Revision: 1.5
 * @date Last Updated: 2/19/2015\n Created: 11/12/2014 7:36:35 PM
 * @brief This file is code to control the ADC on the ATMega328P.
 */

/** @brief The reference voltage for the ATMega328P. */
#define VREF 5.0

#include <avr/io.h>
#include "ADC.h"

/**
 * @brief This function sets up the ADC Pin that the function ADC_Read_Pin
will read.
 * @param ADC_Pin Pin to be set up.\n
 * 0000   | ADC0\n
 * 0001   | ADC1\n
 * 0010   | ADC2\n
 * 0011   | ADC3\n
 * 0100   | ADC4\n
 * 0101   | ADC5\n
 * 0110   | ADC6\n
 * 0111   | ADC7\n
 * 1000   | ADC8\n
 * 1001   | (reserved)\n
 * 1010   | (reserved)\n
 * 1011   | (reserved)\n
 * 1100   | (reserved)\n
 * 1101   | (reserved)\n
 * 1110   | 1.1V (V BG)\n
 * 1111   | 0V (GND)\n
 * @warning This function does not remove the upper 4 bits from the input.
 */
void ADC_Setup_Pin(unsigned char ADC_Pin)
{
    /* Set up which analog input pin to use. */
    ADMUX = ADC_Pin;
    /* Use AVcc as the reference. */
    ADMUX |= (1<<REFS0);
```

```
    /* Clear for 10 bit resolution. */
    ADMUX &= ~(1<<ADLAR);
    /* Set the prescale to 128 for 8Mhz. */
    ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
    /* Enable the ADC. */
    ADCSRA |= (1<<ADEN);
}

/**
 * @brief Reads the selected pin from the function ADC_Setup_Pin.
 * @return The current ADC value from 0 to 1023.
 * @warning ADC_Setup_Pin needs to be ran before this function.
 */
int16_t ADC_Read_Pin(void)
{
    /* Start the ADC conversion. */
    ADCSRA |= (1<<ADSC);
    /* Wait till the ADC complete flag is set. */
    while(ADCSRA&(1<<ADSC));
    /* Gather the lower byte of of the ADC value. */
    uint16_t ADC_Value = ADCL;
    /* Gather the upper byte of of the ADC value. */
    ADC_Value |= (ADCH<<8);
    /* Return the ADC value. */
    return(ADC_Value);
}

/**
 * @brief Converts the ADC value to flat
 * @param ADC_Value The current ADC value from 0 to 1023.
 * @return The current ADC value in voltage from 0 to Vref*1023/1024.
 */
float ADC_Convert_To_Voltage(int16_t ADC_Value)
{
    /* Convert the ADC value to voltage */
    return(((float)ADC_Value*VREF)/1024.0);
}
```

ADC.h

```
/**
 * @file ADC.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 2/16/2015\n Created: 11/12/2014 7:36:35 PM
 * @brief This is the header file for the code that controls the ADC on the
ATMega328P.
 */

#ifndef ADC_H_
#define ADC_H_

#include <stdint.h>

void ADC_Setup_Pin(unsigned char ADC_Pin);
int16_t ADC_Read_Pin(void);
float ADC_Convert_To_Voltage(int16_t ADC_Value);
```

```
#endif /* ADC_H_ */
```

CD4051.c

```
/**
 * @file CD4051.c
 * @author Nicholas Sikkema
 * @version Revision: 1.5
 * @date Last Updated: 1/22/2015\n Created: 10/23/2014 3:23:07 PM
 * @brief This file is code to control the CD4051 multiplexer connected to
the ADC.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif
/** @brief The default pin for the ADC on the ATMega328P. */
#define ADC_DEFAULT_PIN 0

#include <avr/io.h>
#include <util/delay.h>
#include "ADC.h"
#include "CD4051.h"

/**
 * @brief This function is used to initialize the pins used for the
multiplexer
 */
void CD4051_Initialize(void)
{
    /* Initialize ports B1, B2, and B3 as outputs. */
    DDRB|=(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
    /* Initialize the default pin for the ADC. */
    ADC_Setup_Pin(ADC_DEFAULT_PIN);
}

/**
 * @brief This function is to read the selected multiplexer pin
 * Pin 0: Voltage\n
 * Pin 1: Photodiode Front\n
 * Pin 2: Photodiode Right Front\n
 * Pin 3: Photodiode Right Rear\n
 * Pin 4: Photodiode Left Front\n
 * Pin 5: Photodiode Left Rear\n
 * Pin 6: Photodiode Rear\n
 * Pin 7: Pressure Sensor\n
 * @param Multiplexer_Pin Pin to be selected from the multiplexer.
 * @return ADC value for the selected multiplexer pin.
 * @warning This function does not check the input for validity.
 */
int CD4051_Read_Single_Pin(unsigned char Multiplexer_Pin)
{
    /* Clear the lower three bits for PORTB. */
    PORTB &= 0xF8;
    /* Set the multiplexer pin. */
```

```
    PORTB |= Multiplexer_Pin;
    /* Have a delay to allow the switching of the multiplexer. */
    _delay_us(100);
    /* Return the ADC value for the current multiplexer pin. */
    return(ADC_Read_Pin());
}

/**
 * @brief This function is used to calculate the distance for a photodiode.
 * @param Output An array holding ADC values for the selected multiplexer
pins.
 * @param Minimum_Multiplexer_Pin Lowest pin range to be selected from the
multiplexer.
 * @param Maximum_Multiplexer_Pin Highest pin range to be selected from the
multiplexer.
 */
void CD4051_Read_Range(int *Output, unsigned char Minimum_Multiplexer_Pin,
unsigned char Maximum_Multiplexer_Pin)
{
    /* Initialize the index for the array. */
    uint8_t Index = 0;
    /* Loop through the range between the minimum and maximum. */
    for(uint8_t i = Minimum_Multiplexer_Pin; i <= Maximum_Multiplexer_Pin;
i++)
    {
        /* Read the selected pin and save at the current index. */
        Output[Index] = CD4051_Read_Single_Pin(i);
        /* Increment the current index. */
        Index++;
    }
}
```

CD4051.h

```
/**
 * @file CD4051.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/22/2015\n Created: 10/23/2014 3:23:07 PM
 * @brief This file is code to control the CD4051 multiplexer connected to
the ADC.
 */

#ifndef CD4051_H_
#define CD4051_H_

/** @brief The port on the multiplexer that has the current reference
voltage. */
#define Battery_Mux_Port 0
/** @brief Multiplexer port for the front photodiode. */
#define Photodiode_front 1
/** @brief Multiplexer port for the right front photodiode. */
#define Photodiode_rightfront 2
/** @brief Multiplexer port for the right rear photodiode. */
#define Photodiode_rightrear 3
/** @brief Multiplexer port for the left front photodiode. */
#define Photodiode_leftfront 4
```

```
/** @brief Multiplexer port for the left rear photodiode. */
#define Photodiode_leftrear 6
/** @brief The default port for the pressure sensor on the ATMega328P. */
#define Pressure_Sensor_Mux_Port 7

void CD4051_Initialize(void);
void CD4051_Read_Range(int *Output, unsigned char Minimum_Multiplexer_Pin,
unsigned char Maximum_Multiplexer_Pin);
int CD4051_Read_Single_Pin(unsigned char Multiplexer_Pin);

#endif /* CD4051_H_ */
```

I2C_Interface.c

```
/**
 * @file I2C_Interface.c
 * @author Nicholas Sikkema
 * @version Revision: 1.5
 * @date Last Updated: 2/3/2015\n Created: 10/30/2014 3:47:02 PM
 * @brief This file is code to control the TWI_Master.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <stdint.h>
#include <avr/interrupt.h>
#include "I2C_Interface.h"
#include "TWI_Master.h"

/**
 * @brief This function is to initialize the I2C interface.
 */
void I2C_Interface_Initialize()
{
    /* Initialize the TWI master. */
    TWI_Master_Initialise();
    /* Enable global interrupts. */
    sei();
}

/**
 * @brief This function is to send a write I2C message over the I2C bus.
 * @param Device_Address The I2C device address.
 * @param Device_Register The register to write the data.
 * @param Value_To_Write Data to be sent over the bus.
 */
void I2C_Interface_Write(uint8_t Device_Address, uint8_t Device_Register,
uint8_t Value_To_Write)
{
    /* Initialize the message buffer. */
    uint8_t Message_Buff[3];
    /* Set array index 0 to the device address (bits 1-7) and the
TWI_READ_BIT in write mode (bit 0). */
    Message_Buff[0] = (Device_Address<<1)|(FALSE<<TWI_READ_BIT);
```

```
    /* Set array index 1 to the device register you want to write to. */
    Message_Buff[1] = Device_Register;
    /* Set array index 2 to the value you want to write. */
    Message_Buff[2] = Value_To_Write;
    /* Send the write message over the I2C bus. */
    TWI_Start_Transceiver_With_Data(Message_Buff, 3);
}

/**
 * @brief This function is to send and receive a read I2C message over the
I2C bus for a single read.
 * @param Device_Address The I2C device address.
 * @param Device_Register The register to read the data.
 * @return The value from the register wanted.
 */
uint8_t I2C_Interface_Single_Read(uint8_t Device_Address, uint8_t
Device_Register)
{
    /* Initialize a buffer to hold the information from the I2C bus. */
    uint8_t Message_Buff[2];
    /* Set array index 0 to the device address (bits 1-7) and the
TWI_READ_BIT in write mode (bit 0). */
    Message_Buff[0] = (Device_Address<<1)|(FALSE<<TWI_READ_BIT);
    /* Set array index 1 to the device register you want to read from. */
    Message_Buff[1] = Device_Register;
    /* Send the I2C message to indicate specific location and sub. */
    TWI_Start_Transceiver_With_Data(Message_Buff, 2);
    /* Set array index 0 to the device address (bits 1-7) and the
TWI_READ_BIT in read mode (bit 0). */
    Message_Buff[0] = (Device_Address<<1)|(TRUE<<TWI_READ_BIT);
    /* Send a dummy write to set the read pointer for the device. */
    TWI_Start_Transceiver_With_Data(Message_Buff, 2);
    /* Clear the message buffer. */
    Message_Buff[0] = 0;
    Message_Buff[1] = 0;
    /* Retrieve the data from the I2C bus. */
    TWI_Get_Data_From_Transceiver(Message_Buff, 2);
    /* Set the return buffer with the value from the I2C bus. */
    return(Message_Buff[1]);
}

/**
 * @brief This function is to send and receive a read I2C message over the
I2C bus for multiple reads.
 * @param Device_Address The I2C device address.
 * @param Device_Register The register to read the data.\n
 * Note that this function expects that the registers that you are reading
from a continuous.
 * @param Return_Buff An array to store the data from the I2C bus.
 * @param Size_Of_Return_Buff Size of the return buffer array.
 */
void I2C_Interface_Read_Array(uint8_t Device_Address, uint8_t
Device_Register, uint8_t *Return_Buff, uint8_t Size_Of_Return_Buff)
{
    /* Loop for the indexes. */
    for(int i=0; i<Size_Of_Return_Buff; i++)
    {
```

```
        /* Clear the return buffer. */
        Return_Buff[i] = 0;
        /* Set the return buffer with the value from the I2C bus. */
        Return_Buff[i] = I2C_Interface_Single_Read(Device_Address,
Device_Register+i);
    }
}

/**
 * @brief This function is to check if the I2C interface is not busy.
 * @return Boolean true (1) or false (0) for not being busy.
 */
uint8_t I2C_Interface_Not_Busy()
{
    /* Return the opposite of TWI_Transceiver_Busy(). */
    return(!TWI_Transceiver_Busy());
}
```

I2C_Interface.h

```
/**
 * @file I2C_Interface.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/22/2015\n Created: 10/30/2014 4:09:09 PM
 * @brief This file is code to control the TWI_Master.
 */

#ifndef I2C_INTERFACE_H_
#define I2C_INTERFACE_H_

#include <stdint.h>

void I2C_Interface_Initialize(void);
void I2C_Interface_Write(uint8_t Device_Address, uint8_t Device_Register,
uint8_t Value_To_Write);
uint8_t I2C_Interface_Single_Read(uint8_t Device_Address, uint8_t
Device_Register);
void I2C_Interface_Read_Array(uint8_t Device_Address, uint8_t
Device_Register, uint8_t* Return_Buff, uint8_t Size_Of_Return_Buff);
uint8_t I2C_Interface_Not_Busy(void);

#endif /* I2C_INTERFACE_H_ */
```

TWI_Master.c

```
/**
 * Atmel Corporation
 * @file TWI_master.c
 * @author ltwa
 * Commented by: Nicholas Sikkema
 * @version Revision: 1.13
 * @date 24. mai 2004 11:31:20
 * @brief This is a sample driver for the TWI hardware modules.
 * It is interrupt driven. All functionality is controlled through
 * passing information to and from functions.
 */
```

```c
#include <stdint.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "TWI_Master.h"

/* Transceiver buffer */
static uint8_t TWI_buf[TWI_BUFFER_SIZE];
/* Number of bytes to be transmitted. */
static uint8_t TWI_msgSize;//
/* TWI state byte. Default set to TWI_NO_STATE. */
static uint8_t TWI_state = TWI_NO_STATE;//

/* Status byte holding flags. */
static volatile uint8_t lastTransOK = 0;
/* Register for the TWI status. */
static volatile uint8_t TWI_statusReg = 0;

/**
 * @brief Call this function to set up the TWI master to its initial standby
state.
 * @warning Remember to enable interrupts from the main application after
initializing the TWI.
 */
void TWI_Master_Initialise(void)
{
    /* Set bit rate register (Baudrate). Defined in header file. */
    TWBR = TWI_TWBR;
    /* Set TWI prescaler. */
    TWSR = TWI_TWPS;
    /* Default content = SDA released. */
    TWDR = 0xFF;
    /* Enable TWI-interface and release TWI pins, Disable Interrupt, and No
Signal requests. */
    TWCR =
(1<<TWEN)|(0<<TWIE)|(0<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
}

/**
 * @brief Call this function to test if the TWI_ISR is busy transmitting.
 * @return Status of the transceiver.
 */
uint8_t TWI_Transceiver_Busy(void)
{
    /* If the TWI Interrupt is enabled then the transceiver is busy. */
    return(TWCR&(1<<TWIE));
}

/**
 * @brief Call this function to fetch the state information of the previous
operation. The function will hold execution (loop)
 * until the TWI_ISR has completed with the previous operation.
 * @return The TWI state code if there was an error.
 */
uint8_t TWI_Get_State_Info(void)
{
    /* Wait until TWI has completed the transmission. */
    while(TWI_Transceiver_Busy());
```

```
    /* Return the TWI state code if there was an error. */
    return TWI_state;
}

/**
 * @brief Call this function to send a prepared message. The first byte must
contain the slave address and the
 * read/write bit. Consecutive bytes contain the data to be sent, or empty
locations for data to be read
 * from the slave. Also include how many bytes that should be sent/read
including the address byte.
 * The function will hold execution (loop) until the TWI_ISR has completed
with the previous operation,
 * then initialize the next operation and return.
 * @param msg Message buffer to be sent over the I2C bus.
 * @param msgSize Size of the message buffer to be sent over the I2C bus.
 */
void TWI_Start_Transceiver_With_Data(uint8_t *msg, uint8_t msgSize)
{
    /* Wait until TWI is ready for next transmission. */
    while(TWI_Transceiver_Busy());//

    /* Number of data to transmit. */
    TWI_msgSize = msgSize;
    /* Store slave address with R/W setting. */
    TWI_buf[0]  = msg[0];
    /* If it is a write operation, then also copy data. */
    if(!(msg[0]&(TRUE<<TWI_READ_BIT)))
    {
        /* Loop through the indexes of the message buffer. */
        for(uint8_t i = 1; i<msgSize; i++)
        {
            /* Copy index i of the msg buffer to index i of TWI_buf*/
            TWI_buf[i] = msg[i];
        }
    }

    /* Set the status register to zero*/
    TWI_statusReg = 0;
    /* Reset the TWI state. */
    TWI_state = TWI_NO_STATE;
    /* Set the TWCR to enable the TWI interface, enable TWI interrupt and
clear the flag, and initiate the start condition. */
    TWCR =
(1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
}

/**
 * @brief Call this function to resend the last message. The driver will
reuse the data previously put in the transceiver buffers.
 * The function will hold execution (loop) until the TWI_ISR has completed
with the previous operation,
 * then initialize the next operation and return.
 */
void TWI_Start_Transceiver(void)
{
    /* Wait until TWI is ready for next transmission. */
```

```
    while(TWI_Transceiver_Busy());
    /* Set the status register to zero*/
    TWI_statusReg = 0;
    /* Reset the TWI state. */
    TWI_state = TWI_NO_STATE;
    /* Set the TWCR to enable the TWI interface, enable TWI interrupt and
clear the flag, and initiate the start condition. */
    TWCR =
(1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
}

/**
 * @brief Call this function to read out the requested data from the TWI
transceiver buffer. I.e. first call
 * TWI_Start_Transceiver to send a request for data to the slave. Then Run
this function to collect the
 * data when they have arrived. Include a pointer to where to place the data
and the number of bytes
 * requested (including the address field) in the function call. The function
will hold execution (loop)
 * until the TWI_ISR has completed with the previous operation, before
reading out the data and returning.
 * @param msg Message buffer to be received from the I2C bus.
 * @param msgSize Size of the message buffer to be received from the I2C bus.
 * @return
 */
uint8_t TWI_Get_Data_From_Transceiver(uint8_t *msg, uint8_t msgSize)
{
    /* Wait until TWI is ready for next transmission. */
    while(TWI_Transceiver_Busy());
    /* Last transmission competed successfully. */
    if(lastTransOK)
    {
        /* Loop through the indexes of the message buffer. */
        for(uint8_t i = 0; i<msgSize; i++)
        {
            /* Copy index i of the TWI_buf buffer to index i of msg. */
            msg[i] = TWI_buf[i];
        }
    }
    /* Return TWI error code if there was an error in the previous
transmission the function. */
    return lastTransOK;
}

/**
 * @brief This function is the Interrupt Service Routine (ISR), and called
when the TWI interrupt is triggered;
 * that is whenever a TWI event has occurred. This function should not be
called directly from the main
 * application.
 */
ISR(TWI_vect)
{
    /* Pointer index for the TWI buffer. */
    static uint8_t TWI_bufPtr;
```

```
    switch(TWSR)
    {
        /* Start has been transmitted. */
        case TWI_START:
        /* Repeated START has been transmitted. */
        case TWI_REP_START:
            /* Set buffer pointer to the TWI Address location. */
            TWI_bufPtr = 0;
        /* SLA+W has been transmitted and ACK received. */
        case TWI_MTX_ADR_ACK:
        /* Data byte has been transmitted and ACK received. */
        case TWI_MTX_DATA_ACK:
            /* Check if the pointer is less than last bit. */
            if(TWI_bufPtr < TWI_msgSize)
            {
                /* Update the TWDR with the transceiver buffer information
and increment the pointer index*/
                TWDR = TWI_buf[TWI_bufPtr++];
                /* Set the TWCR to enable the TWI interface and enable TWI
interrupt and clear the flag to send byte. */
                TWCR =
(1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
            }
            else
            {
                /* Set status bits to completed successfully. */
                lastTransOK = TRUE;
                /* Set the TWCR to enable the TWI interface, disable TWI
interrupt and clear the flag to send byte, and initiate a stop condition. */
                TWCR =
(1<<TWEN)|(0<<TWIE)|(1<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|(0<<TWWC);
            }
            break;
        /* Data byte has been received and acknowledge transmitted. */
        case TWI_MRX_DATA_ACK:
            /* Update the transceiver buffer with the TWDR information. */
            TWI_buf[TWI_bufPtr++] = TWDR;
        /* SLA+R has been transmitted and acknowledge received. */
        case TWI_MRX_ADR_ACK:
            /* Check if the pointer is less than the second to last bit. */
            if(TWI_bufPtr < (TWI_msgSize-1))
            {
                /* Set the TWCR to enable the TWI interface, enable TWI
Interrupt and clear the flag to read next byte, and send acknowledge after
reception. */
                TWCR =
(1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
            }
            else
            {
                /* Set the TWCR to enable the TWI interface, enable TWI
Interrupt and clear the flag to read next byte, and send negative-acknowledge
after reception. */
                TWCR =
(1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
            }
            break;
```

```
        /* Data byte has been received and negative-acknowledge transmitted.
*/
        case TWI_MRX_DATA_NACK:
            /* Update the transceiver buffer with the TWDR information. */
            TWI_buf[TWI_bufPtr] = TWDR;
            /* Set status bits to completed successfully. */
            lastTransOK = TRUE;
            /* Set the TWCR to enable the TWI interface, disable TWI
Interrupt and clear the flag, and initiate a stop condition. */
            TWCR =
(1<<TWEN)|(0<<TWIE)|(1<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|(0<<TWWC);
            break;
        /* Arbitration lost on the I2C bus. */
        case TWI_ARB_LOST:
            /* Set the TWCR to enable the TWI interface, enable TWI Interrupt
and clear the flag, and initiate a restart condition. */
            TWCR =
(1<<TWEN)|(1<<TWIE)|(1<<TWINT)|(0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|(0<<TWWC);
            break;
        /* Bus error due to an illegal start or stop condition. */
        case TWI_BUS_ERROR:
        /* SLA+W has been transmitted and negative-acknowledge received. */
        case TWI_MTX_ADR_NACK:
        /* SLA+R has been transmitted and negative-acknowledge received. */
        case TWI_MRX_ADR_NACK:
        /* Data byte has been transmitted and negative-acknowledge received.
*/
        case TWI_MTX_DATA_NACK:
        default:
            /* Store TWSR and automatically clears no Errors bit. */
            TWI_state = TWSR;
            /* Set the TWCR to enable the TWI interface and release TWI pins,
disable TWI Interrupt flag, and initiate a stop condition. */
            TWCR =
(1<<TWEN)|(0<<TWIE)|(1<<TWINT)|(0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|(0<<TWWC);
    }
}
```

TWI_Master.h

```
/**
 * Atmel Corporation
 * @file TWI_master.h
 * @author ltwa
 * @version Revision: 1.13
 * @date 24. mai 2004 11:31:22
 * @brief Header file for TWI_master.c.
 * Include this file in the application.
 */

#ifndef TWI_MASTER_H_
#define TWI_MASTER_H_

#include <stdint.h>

/** @brief Set this to the largest message size that will be sent including
address byte. */
```

```
#define TWI_BUFFER_SIZE 16
/** @brief TWI_BITRATE in kHz. */
#define TWI_BITRATE 400

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

/** @brief TWI Bit rate Register setting. */
#define TWI_PSBR (F_CPU/2000/TWI_BITRATE-8)

#if TWI_PSBR > 0x3FC0
   #define TWI_TWPS 0x03
   #define TWI_TWBR 0xFF
   #define MESSAGE 'a'
#elif TWI_PSBR > 0x0FF0
   #define TWI_TWPS 0x03
   #define TWI_TWBR TWI_PSBR/64
   #define MESSAGE 'b'
#elif TWI_PSBR > 0x03FC
   #define TWI_TWPS 0x02
   #define TWI_TWBR TWI_PSBR/16
   #define MESSAGE 'c'
#elif TWI_PSBR > 0x00FF
   #define TWI_TWPS 0x01
   #define TWI_TWBR TWI_PSBR/4
   #define MESSAGE 'd'
#else
   #define TWI_TWPS 0x00
   #define TWI_TWBR TWI_PSBR
   #define MESSAGE 'e'
#endif

/** @brief Bit position for R/W bit in "address byte". */
#define TWI_READ_BIT 0
/** @brief Bit position for LSB of the slave address bits in the initial
byte. */
#define TWI_ADR_BITS 1

/** @brief Boolean equivalent for TRUE. */
#define TRUE 1
/** @brief Boolean equivalent for FALSE. */
#define FALSE 0

/* General TWI Master status codes. */
/** @brief START has been transmitted. */
#define TWI_START 0x08
/** @brief Repeated START has been transmitted. */
#define TWI_REP_START 0x10
/** @brief Arbitration lost. */
#define TWI_ARB_LOST 0x38

/* TWI Master Transmitter status codes. */
/** @brief SLA+W has been tramsmitted and ACK received. */
#define TWI_MTX_ADR_ACK 0x18
/** @brief SLA+W has been tramsmitted and NACK received. */
```

```
#define TWI_MTX_ADR_NACK 0x20
/** @brief Data byte has been tramsmitted and ACK received. */
#define TWI_MTX_DATA_ACK 0x28
/** @brief Data byte has been tramsmitted and NACK received. */
#define TWI_MTX_DATA_NACK 0x30

/* TWI Master Receiver status codes. */
/** @brief SLA+R has been tramsmitted and ACK received. */
#define TWI_MRX_ADR_ACK 0x40
/** @brief SLA+R has been tramsmitted and NACK received. */
#define TWI_MRX_ADR_NACK 0x48
/** @brief Data byte has been received and ACK tramsmitted. */
#define TWI_MRX_DATA_ACK 0x50
/** @brief Data byte has been received and NACK tramsmitted. */
#define TWI_MRX_DATA_NACK 0x58

/* TWI Slave Transmitter status codes. */
/** @brief Own SLA+R has been received and ACK has been returned. */
#define TWI_STX_ADR_ACK 0xA8
/** @brief Arbitration lost in SLA+R/W as Master, own SLA+R has been
received, and ACK has been returned. */
#define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0
/** @brief Data byte in TWDR has been transmitted and ACK has been received.
*/
#define TWI_STX_DATA_ACK 0xB8
/** @brief Data byte in TWDR has been transmitted and NOT ACK has been
received. */
#define TWI_STX_DATA_NACK 0xC0
/** @brief Last data byte in TWDR has been transmitted (TWEA = '0') ACK has
been received. */
#define TWI_STX_DATA_ACK_LAST_BYTE 0xC8

/* TWI Slave Receiver status codes. */
/** @brief Own SLA+W has been received ACK has been returned. */
#define TWI_SRX_ADR_ACK 0x60
/** @brief Arbitration lost in SLA+R/W as Master, own SLA+W has been
received, and ACK has been returned. */
#define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68
/** @brief General call address has been received and ACK has been returned.
*/
#define TWI_SRX_GEN_ACK 0x70
/** @brief Arbitration lost in SLA+R/W as Master, General call address has
been received, and ACK has been returned. */
#define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78
/** @brief Previously addressed with own SLA+W, data has been received, and
ACK has been returned. */
#define TWI_SRX_ADR_DATA_ACK 0x80
/** @brief Previously addressed with own SLA+W, data has been received, and
NOT ACK has been returned. */
#define TWI_SRX_ADR_DATA_NACK 0x88
/** @brief Previously addressed with general call, data has been received,
and ACK has been returned. */
#define TWI_SRX_GEN_DATA_ACK 0x90
/** @brief Previously addressed with general call, data has been received,
and NOT ACK has been returned. */
#define TWI_SRX_GEN_DATA_NACK 0x98
```

```
/** @brief A STOP condition or repeated START condition has been received
while still addressed as Slave. */
#define TWI_SRX_STOP_RESTART 0xA0

/* TWI Miscellaneous status codes. */
/** @brief No relevant state information available. */
#define TWI_NO_STATE 0xF8
/** @brief Bus error due to an illegal START or STOP condition. */
#define TWI_BUS_ERROR 0x00

void TWI_Master_Initialise(void);
uint8_t TWI_Transceiver_Busy(void);
uint8_t TWI_Get_State_Info(void);
void TWI_Start_Transceiver_With_Data(uint8_t *msg, uint8_t msgSize);
void TWI_Start_Transceiver(void);
uint8_t TWI_Get_Data_From_Transceiver(uint8_t *msg, uint8_t msgSize);

#endif
```

USART.c

```
/**
 * @file USART.c
 * @author Ryan Lipski, Nick Sikkema
 * @version Revision: 2.0
 * @date Last Updated: 4/21/2015\n Created: 11/12/2014 7:36:35 PM
 * @brief This file is code to control the USART on the ATMega328P.
 */

/** @brief The current clock speed for the microcontroller. */
#define FOSC 16000000
/** @brief The desired baud rate for the USART connection. */
#define BAUD 76800
/** @brief The desired baud rate for the USART connection. */
#define MYUBRR (FOSC/(16UL*BAUD)-1)
/** @brief The desired precision for the conversion between float and string.
*/
#define PRECISION 3

#include <avr/io.h>
#include "USART.h"

/**
 * @brief Initializes USART on the ATMega328P.
 */
void USART_Initialize(void)
{
    /* Set baud rate. */
    UBRR0H = (unsigned char)(MYUBRR>>8);
    UBRR0L = (unsigned char)MYUBRR;
    /* Clear UCSR0A. This is to make sure that the USART is in single
asynchronous mode. */
    UCSR0A = 0x00;
    /* Enable receiver and transmitter. */
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);
    /* Set frame format: 8 data and 1 stop bit. */
    UCSR0C = (3<<UCSZ00);
```

```
}

/**
 * @brief Sends a single byte using USART.
 * @param data Character that is to be sent.
 */
void USART_Send_Byte(char data)
{
    /* Wait for empty transmit buffer. */
    while(!(UCSR0A & (1<<UDRE0)));
    /* Put data into buffer, sends the data. */
    UDR0 = data;
}

/**
 * @brief Receives a single byte using USART.
 * @return Character that is received.
 */
unsigned char USART_Receive_Byte(void)
{
    /* Wait for data to be received. */
    while(!(UCSR0A & (1<<RXC0)));
    /* Get and return received data from buffer. */
    return(UDR0);
}

/**
 * @brief Sends a c style string using USART.
 * @param string The c style string that is to be sent.
 */
void USART_Send_String(char* string)
{
    /* Loop through the string pointer. */
    for(char* pointer=string; *pointer!=0x00; pointer++) {
        /* Send the character. */
        USART_Send_Byte(*pointer);
    }
}

/**
 * @brief Receives a c style string using USART.
 * @param string The c style string that is to be received.
 * @return The number of characters that are received.
 */
int USART_Read_String(char* string)
{
    /* A variable to keep track of the string length. */
    int count=0;
    while (1) {
        /** Receive the information from the keyboard and save it in a
temporary variable. */
        char collector=USART_Receive_Byte();
        /* Break if the enter key is pressed. Check to see if the carriage
return and line feed is sent. */
        if (collector=='\n'||collector=='\r'){
            break;
        }
```

```
        /* Update the string with the byte received. */
        *string++=collector;
        /* Increase the length count. */
        count++;
    }
    /* terminate the char array string with 0x00/ */
    *string=0x00;
    /* Return the length of the string. */
    return(count);
}

/**
 * @brief Receives a c style string using USART, then send the c style
string.
 * @param string The c style string that is to be received.
 * @return The number of characters that are received.
 */
int USART_Read_String_With_Echo(char* string)
{
    /* A variable to keep track of the string length. */
    int count=0;
    while(1) {
        /** Receive the information from the keyboard and save it in a
temporary variable. */
        char collector=USART_Receive_Byte();
        /* Send back the information just received to the screen. */
        USART_Send_Byte(collector);
        /* Break if the enter key is pressed. Check to see if the carriage
return and line feed is sent. */
        if (collector=='\n'||collector=='\r'){
            break;
        }
        /* Update the string with the byte received. */
        *string++=collector;
        /* Increase the length count. */
        count++;
    }
    /* terminate the char array string with 0x00/ */
    *string=0x00;
    /* Return the length of the string. */
    return(count);
}

/**
 * @brief Converts a given floating point value to a string.
 * @param Input_Value The floating point value to be converted.
 * @param Buffer The buffer pointer for the character array.
 * @return The floating point value in string form.
 */
char* USART_ftoa(float Input_Value, char* Buffer)
{
    /* Initialize a index variable. */
    unsigned int i = 0;
    /* Loop through the indexes and reinitialize the buffer. */
    for(i = 0; i<FLOATING_POINT_BUFFER_SIZE; i++)
    {
```

```
        /* Set the buffer index to NULL. */
        Buffer[i] = 0;
    }
    /* Copy the Input_Value to a local variable. Removing the decimal value.
*/
    int Temp_Value = Input_Value;
    /* Copy the Input_Value to a local variable. */
    float Temp_Input_Value = Input_Value;
    /* Reset the index pointer. */
    i = 0;
    /* Check to see if the Temp_Input_Value is negative. */
    if(Temp_Input_Value<0)
    {
        /* If the Input_Value is negative then flip sign. */
        Temp_Value = -Temp_Value;
        Temp_Input_Value = -Temp_Input_Value;
        /* Add a negative sign*/
        Buffer[i++] = '-';
    }
    /* Process individual digits */
    if (Temp_Value==0)
    {
        /* Add zero to the array. */
        Buffer[i++] = 48;
    }
    else
    {
        /* Copy the integer portion of the Input_Value to the Buffer. */
        while (Temp_Value != 0)
        {
            /* Add the current digit to the array. */
            Buffer[i++] = (char)(Temp_Value%10+48);
            /* Remove the last digit. */
            Temp_Value = Temp_Value/10;
        }
    }
    /* Add the decimal place. */
    Buffer[i++] = '.';
    /* Reset the Temp_Value. */
    Temp_Value = (int)Temp_Input_Value;
    /* Initialize a precision counter. */
    int Precision_Count = 0;
    do {
        /* Update the floating portion by subtracting the integer portion. */
        Temp_Input_Value -= Temp_Value;
        /* Update the integer portion by shifting the decimal place and
remove the decimal places. */
        Temp_Value = Temp_Input_Value*10;
        /* Append the new integer value to the Buffer. */
        Buffer[i++] = (char)(Temp_Value+48);
        /* Update the precision count. */
        Precision_Count++;
        /* Update the floating portion by shifting the decimal place. */
        Temp_Input_Value *= 10;
        /* Check to see if the index is greater than the buffer size or that
the precision is good enough. */
    } while(i < FLOATING_POINT_BUFFER_SIZE && Precision_Count < PRECISION);
```

```
    /* Return the buffer pointer. */
    return(Buffer);
}
```

USART.h

```
/**
 * @file USART.h
 * @author Ryan Lipski, Nick Sikkema
 * @version Revision: 1.5
 * @date Last Updated: 3/20/2015\n Created: 11/12/2014 7:36:35 PM
 * @brief This is the header file to control the USART on the ATMega328P.
 */

#ifndef USART_H_
#define USART_H_

/** @brief Max buffer size for the conversion between float and string. */
#define FLOATING_POINT_BUFFER_SIZE 15

void USART_Initialize(void);
void USART_Send_Byte(char data);
unsigned char USART_Receive_Byte(void);
void USART_Send_String(char* string);
int USART_Read_String(char* string);
int USART_Read_String_With_Echo(char* string);
char * USART_ftoa(float Input_Value, char *Buffer);

#endif /* USART_H_ */
```

Interrupt.c

```
/**
 * @file Interrupt.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/29/2015\n Created: 1/27/2015 4:38:39 PM
 * @brief This file is code for the interrupts on the ATMega328P.
 */

#include <avr/interrupt.h>
#include "../../Algorithms/Motor_Control/Motor_Control.h"
#include "../../Algorithms/Navigation/Swarming.h"
#include "../../Algorithms/State_Machine/State_Machine.h"
#include "../../Microcontroller/Interrupt/Interrupt.h"

/**
 * @brief Interrupt for timer 0 compare A vector, which is used to update the
motor control value.
 */
ISR(TIMER0_COMPA_vect)
{
    /* Enable the flag to update the motor control values. */
    Motor_Control_Enable_Flag();
}

/**
```

```
 * @brief Interrupt for timer 1 compare A vector, which is used to update the
swarming value.
 */
ISR(TIMER1_COMPA_vect)
{
    /* Enable the flag to update the swarming values. */
    Swarming_Enable_Flag();
}

/**
 * @brief Interrupt for timer 2 compare A vector, which is used to update the
state machine.
 */
ISR(TIMER2_COMPA_vect)
{
    /* Enable the flag to update the state machine. */
    State_Machine_Enable_Flag();
}
```

Interrupt.h

```
/**
 * @file Interrupt.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/29/2015\n Created: 1/27/2015 4:28:45 PM
 * @brief This file is code for the interrupts on the ATMega328P.
 */

#ifndef INTERRUPT_H_
#define INTERRUPT_H_

#endif /* INTERRUPT_H_ */
```

Timer.c

```
/**
 * @file Timer.c
 * @author Nicholas Sikkema
 * @version Revision: 2.0
 * @date Last Updated: 4/21/2015 PM\n Created: 11/12/2014
 * @brief This file is code to control the Timers on the ATMega328P.
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include "Timer.h"

/**
 * @brief Initializes Timer0 with a period of 20ms.
 */
inline void Timer0_Initialize ()
{
    /* Set Timer0 initial counter value to 0. */
    TCNT0 = 0x00;
    /* Set timer period of 12.5 ms. */
    OCR0A = 0xC2;
    /* Set the output to be in CTC mode. */
```

```
    TCCR0A |= (1<<WGM01);
    /* Set the timer prescaler to 1024. */
    TCCR0B |= (1<<CS02)|(1<<CS00);
}

/**
 * @brief Enable Timer0 interrupt flag.
 */
void Timer0_Enable()
{
    /* Set the Timer0A Interrupt flag to 1. */
    TIMSK0 |= (1<<OCIE0A);
}

/**
 * @brief Disable Timer0 interrupt flag.
 */
void Timer0_Disable()
{
    /* Set the Timer0A Interrupt flag to 0. */
    TIMSK0 &= ~(1<<OCIE0A);
}

/**
 * @brief Initializes Timer1 as an interrupt at a period of ?.
 */
inline void Timer1_Initialize ()
{
    /* Set Timer0 initial counter value to 0. */
    TCNT1 = 0x00;
    /* Set timer period of 25 ms. */
    OCR1A = 0x0186;
    /* Set the output to be in CTC mode. */
    TCCR1A |= (1<<WGM12);
    /* Set the timer prescaler to 1024. */
    TCCR1B |= (1<<CS12)|(1<<CS10);
}

/**
 * @brief Enable Timer1 interrupt flag.
 */
void Timer1_Enable()
{
    /* Set the Timer1A Interrupt flag to 1. */
    TIMSK1 |= (1<<OCIE1A);
}

/**
 * @brief Disable Timer1 interrupt flag.
 */
void Timer1_Disable()
{
    /* Set the Timer1A Interrupt flag to 0. */
    TIMSK1 &= ~(1<<OCIE1A);
}
```

```
/**
 * @brief Initializes Timer2 with a period of ?.
 */
inline void Timer2_Initialize ()
{
    /* Set Timer0 initial counter value to 0. */
    TCNT2 = 0x00;
    /* Set timer period of 12.5 ms. */
    OCR2A = 0xC2;
    /* Set the output to be in CTC mode. */
    TCCR2A |= (1<<WGM21);
    /* Set the timer prescaler to 1024. */
    TCCR2B |= (1<<CS22)|(1<<CS20);
}

/**
 * @brief Enable Timer2 interrupt flag.
 */
void Timer2_Enable()
{
    /* Set the Timer2A interrupt flag to 1. */
    TIMSK2 |= (1<<OCIE2A);
}

/**
 * @brief Disable Timer2 interrupt flag.
 */
void Timer2_Disable()
{
    /* Set the Timer2A interrupt flag to 0. */
    TIMSK2 &= ~(1<<OCIE2A);
}

/**
 * @brief Initializes the timers that are used in the project.
 */
void Timer_Initialize(void)
{
    /* Initialize the timers. */
    Timer0_Initialize();
    Timer1_Initialize();
    Timer2_Initialize();
    /* Enable global interrupts. */
    sei();
}
```

Timer.h

```
/**
 * @file Timer.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 2/12/2015\n Created:
 * @brief This file is code to control the Timers on the ATMega328P.
 */

#ifndef Timer_H_
```

```
#define Timer_H_

void Timer_Initialize(void);
void Timer0_Enable(void);
void Timer0_Disable(void);
void Timer1_Enable(void);
void Timer1_Disable(void);
void Timer2_Enable(void);
void Timer2_Disable(void);

#endif
```

DRV8830.c

```
/**
 * @file DRV8830.c
 * @author Nicholas Sikkema
 * @version Revision: 2.0
 * @date Last Updated: 4/21/2015\n Created: 11/8/2014 3:54:57 PM
 * @brief This file is code to read control the h-bridges.
 */

/** @brief The conversion factor used between the floating point value and
the value for the h-bridge. */
#define H_Bridge_Voltage_Conversion 12.5
/** @brief The h-bridge value that is equivalent to Motor_Max_Voltage. */
#define H_Bridge_Max_Voltage_VSET
(uint8_t)(H_Bridge_Voltage_Conversion*Motor_Max_Voltage)
/** @brief The minimum voltage that the h-bridge will allow. */
#define H_Bridge_Min_Voltage 0.48
/** @brief The h-bridge value that is equivalent to Motor_Max_Voltage. */
#define H_Bridge_Min_Voltage_VSET
(uint8_t)(H_Bridge_Voltage_Conversion*H_Bridge_Min_Voltage)
/** @brief The clear fault bit for the h-bridge. */
#define Clear_Fault_Bit 7
/** @brief Maximum amount of times allowed to fault before sending that the
motor failed. */
#define Motor_Fault_Count_Max 5
/** @brief Maximum voltage jump allowed for a motor. */
#define Motor_Max_Jump 25

/** @brief I2C h-bridge addresses for the motors. */
enum Motor_Adress
{
    Motor_Address_X = 0x60,
    Motor_Address_Y = 0x62,
    Motor_Address_Z = 0x66
};

/** @brief The register addresses for the DRV8830. */
enum DRV8830_Registers
{
    DRV8830_Control = 0x00,
    DRV8830_Fault =  0x01,
};

/** @brief The list of motor modes for the DRV8830. */
```

```
enum DRV8830_Modes
{
    DRV8830_Standby = 0,
    DRV8830_Reverse =  1,
    DRV8830_Forward = 2,
    DRV8830_Stop = 3,
};

#include <stdio.h>
#include <avr/io.h>
#include "../../Master_Build_Control.h"
#include "../../Microcontroller/Communication/I2C_Interface.h"
#include "../../Microcontroller/Communication/USART.h"
#include "DRV8830.h"

/* Static variables to keep track of the last voltage that was sent to the h-
bridge. */
static uint8_t Voltage_Value_X_Old = 0;
static uint8_t Voltage_Value_Y_Old = 0;
static uint8_t Voltage_Value_Z_Old = 0;

/* Static variables to keep track of the last mode that was sent to the h-
bridge. */
static uint8_t Mode_X_Old = 0;
static uint8_t Mode_Y_Old = 0;
static uint8_t Mode_Z_Old = 0;

/* Static variables to keep track of the motors that are trying to start-up.
*/
static uint8_t Motor_X_Start_Up_Flag = 0;
static uint8_t Motor_Y_Start_Up_Flag = 0;
static uint8_t Motor_Z_Start_Up_Flag = 0;

/**
 * @brief This function converts a desired voltage to the equivalent voltage
value for the h-bridge.
 * @param Voltage The desired voltage for the motor to be at.
 * @return The equivalent voltage value for the h-bridge.
 */
inline uint8_t DRV8830_Voltage_To_Int(float Voltage)
{
    /* Check to see if the input voltage is greater than the max rating of
the motors that we are using. */
    if(Voltage>Motor_Max_Voltage)
    {
        /* To avoid burning up the motor if the input voltage is above the
max voltage then return */
        return(H_Bridge_Max_Voltage_VSET);
    }
    /* Check to see if the input voltage is less than minimum voltage for the
h-bridge. */
    else if(Voltage<H_Bridge_Min_Voltage)
    {
        /* If the input voltage is below the h-bridge minimum voltage, the
only thing available is zero. */
        return(0);
    }
```

```
    /* Convert the voltage to a usable 8 bit value. */
    else
    {
        /* Return the converted h-bridge voltage value. */
        return((uint8_t)(Voltage*H_Bridge_Voltage_Conversion));
    }
}

/**
 * @brief This function checks the desired equivalent voltage value.
 * Checking against the previous mode and voltage, to make sure of three
conditions.
 * First is that the new voltage is not significantly greater than the
previous voltage.
 * Second is to set the flags for the start-up of the motor. This will be
used to make sure that
 * not more one motor is being started up at the same time.
 * Third is to prevent the motors from switching polarity too quickly.
 * @param New_Voltage The desired equivalent voltage value of the h-bridge.
 * Note that the function updates this variable with a new voltage if it
matches on of the three conditions.
 * @param New_Mode The desired mode of the h-bridge.
 * Note that the function updates this variable with a new mode if it matches
on of the three conditions.
 * @param Old_Voltage The current equivalent voltage value of the h-bridge.
 * @param Old_Mode The mode of the h-bridge.
 * @return The start-up flag information for the motor.
 */
inline uint8_t DRV8830_Jump_Check(uint8_t *New_Voltage, uint8_t *New_Mode,
uint8_t Old_Voltage, uint8_t Old_Mode)
{
    /* Initialize the variable that holds the start-up flag information. */
    uint8_t Return_Value = 0;
    /* Check the current mode of the h-bridge. */
    switch (Old_Mode)
    {
        /* The current mode of the h-bridge is reverse. */
        case DRV8830_Reverse:
            /* Check the desired mode of the h-bridge. */
            switch(*New_Mode)
            {
                /* The desired mode is reverse. */
                case DRV8830_Reverse:
                    /* Limit a positive voltage jump. */
                    if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
                    {
                        /* Increase the voltage by the maximum jump. */
                        *New_Voltage = Old_Voltage+Motor_Max_Jump;
                    }
                    /* Limit a negative voltage jump. */
                    else if(Old_Voltage-*New_Voltage>Motor_Max_Jump)
                    {
                        /* Decrease the voltage by the maximum jump. */
                        *New_Voltage = Old_Voltage-Motor_Max_Jump;
                    }
                    break;
```

```
                /* The desired mode is forward. */
                case DRV8830_Forward:
                    /* Check to see if the motor voltage can be decreased
without going to a complete stop. */
                    if(Old_Voltage>=Motor_Max_Jump+H_Bridge_Min_Voltage_VSET)
                    {
                        /* Decrease the motor voltage. */
                        *New_Voltage = Old_Voltage-Motor_Max_Jump;
                        *New_Mode = Old_Mode;
                    }
                    else
                    {
                        /* Stop the motor by setting the mode to stop and
voltage to 0. */
                        *New_Voltage = 0;
                        *New_Mode = DRV8830_Stop;
                    }
                    break;
            }
            break;
        /* The current mode of the h-bridge is forward. */
        case DRV8830_Forward:
            /* Check the desired mode of the h-bridge. */
            switch(*New_Mode)
            {
                /* The desired mode is reverse. */
                case DRV8830_Reverse:
                    /* Check to see if the motor voltage can be decreased
without going to a complete stop. */
                    if(Old_Voltage>=Motor_Max_Jump+H_Bridge_Min_Voltage_VSET)
                    {
                        /* Decrease the motor voltage. */
                        *New_Voltage = Old_Voltage-Motor_Max_Jump;
                        *New_Mode = Old_Mode;
                    }
                    else
                    {
                        /* Stop the motor by setting the mode to stop and
voltage to 0. */
                        *New_Voltage = 0;
                        *New_Mode = DRV8830_Stop;
                    }
                    break;
                /* The desired mode is forward. */
                case DRV8830_Forward:
                    /* Limit a positive voltage jump. */
                    if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
                    {
                        /* Increase the voltage by the maximum jump. */
                        *New_Voltage = Old_Voltage+Motor_Max_Jump;
                    }
                    /* Limit a negative voltage jump. */
                    else if(Old_Voltage-*New_Voltage>Motor_Max_Jump)
                    {
                        /* Decrease the voltage by the maximum jump. */
                        *New_Voltage = Old_Voltage-Motor_Max_Jump;
                    }
```

```
                    break;
                }
                break;
            /* The current mode of the h-bridge is stop. */
            default:
                /* Check the desired mode of the h-bridge. */
                switch(*New_Mode)
                {
                    /* The desired mode is reverse. */
                    case DRV8830_Reverse:
                        /* Set the start-up flag. */
                        Return_Value = 1;
                        /* Limit a positive voltage jump. */
                        if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
                        {
                            /* Increase the voltage by the maximum jump. */
                            *New_Voltage = Old_Voltage+Motor_Max_Jump;
                        }
                        break;
                    /* The desired mode is forward. */
                    case DRV8830_Forward:
                        /* Set the start-up flag. */
                        Return_Value = 1;
                        /* Limit a positive voltage jump. */
                        if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
                        {
                            /* Increase the voltage by the maximum jump. */
                            *New_Voltage = Old_Voltage+Motor_Max_Jump;
                        }
                        break;
                }
                break;
        }
        /* Return the start-up flag information. */
        return(Return_Value);
}

/**
 * @brief This function uses the a desired voltage to determine the mode for
the h-bridge.
 * @param Voltage The desired voltage for the motor to be at.
 * @return The new mode for the h-bridge.
 */
inline uint8_t DRV8830_Mode(float *Voltage)
{
    /* Initialize the variable that holds mode information. */
    uint8_t Return_Value = DRV8830_Stop;
    /* Check to see if the voltage is positive or negative. */
    if(*Voltage<0.0)
    {
        /* Invert the voltage if the voltage is negative. */
        *Voltage*=-1;
        /* Check to see if the voltage is above or equal to the minimum
voltage. */
        if(*Voltage>=H_Bridge_Min_Voltage)
        {
```

```
            /* Set the motor mode to reverse. */
            Return_Value = DRV8830_Reverse;
        }
    }
    else
    {
        /* Check to see if the voltage is above or equal to the minimum
voltage. */
        if(*Voltage>=H_Bridge_Min_Voltage)
        {
            /* Set the motor mode to forward. */
            Return_Value = DRV8830_Forward;
        }
    }
    /* Return the mode information. */
    return(Return_Value);
}

/**
 * @brief This function checks whether or not a motor should be updated.
 * Making sure that there are no motors that are turning on at the same time.
 * @param Motor_Address The motor address of the motor to be updated.
 * @return Whether or not the motor should update.
 */
uint8_t DRV8830_Start_Up_Flag_Control(uint8_t Motor_Address)
{
    /* Initialize the variable that holds the information needed for updating
the motor. */
    uint8_t Return_Value = 0;
    /* Variables to keep track of which motor is starting up. */
    static uint8_t Start_Up_X = 0;
    static uint8_t Start_Up_Y = 0;
    static uint8_t Start_Up_Z = 0;
    /* Check the motor address. */
    switch(Motor_Address)
    {
        /* Check the information for the X motor. */
        case Motor_Address_X:
            /* Check if the routine wants to start-up the motor. */
            if(Motor_X_Start_Up_Flag)
            {
                /* Check if another motor is starting up. */
                if(Start_Up_Y||Start_Up_Z)
                {
                    /* Keep the motor off due to another motor starting. */
                    Start_Up_X = 0;
                }
                else
                {
                    /* Allow the motor to update. */
                    Return_Value = 1;
                    /* Start-up the motor. */
                    Start_Up_X = 1;
                }
            }
            else
            {
```

```
                /* Normal operation of the motor. */
                Return_Value = 1;
                /* Clear the startup flag. */
                Start_Up_X = 0;
            }
            break;
        /* Check the information for the Y motor. */
        case Motor_Address_Y:
            /* Check if the routine wants to start-up the motor. */
            if(Motor_Y_Start_Up_Flag)
            {
                /* Check if another motor is starting up. */
                if(Start_Up_X||Start_Up_Z)
                {
                    /* Keep the motor off due to another motor starting and
clear the start-up variable. */
                    Start_Up_Y = 0;
                }
                else
                {
                    /* Allow the motor to update. */
                    Return_Value = 1;
                    /* Start-up the motor. */
                    Start_Up_Y = 1;
                }
            }
            else
            {
                /* Normal operation of the motor. */
                Return_Value = 1;
                /* Clear the startup flag. */
                Start_Up_Y = 0;
            }
            break;
        /* Check the information for the Z motor. */
        case Motor_Address_Z:
            /* Check if the routine wants to start-up the motor. */
            if(Motor_Z_Start_Up_Flag)
            {
                /* Check if another motor is starting up. */
                if(Start_Up_X||Start_Up_Y)
                {
                    /* Keep the motor off due to another motor starting and
clear the start-up variable. */
                    Start_Up_Z = 0;
                }
                else
                {
                    /* Allow the motor to update. */
                    Return_Value = 1;
                    /* Start-up the motor. */
                    Start_Up_Z = 1;
                }
            }
            else
            {
                /* Normal operation of the motor. */
```

```
                Return_Value = 1;
                /* Clear the startup flag. */
                Start_Up_Z = 0;
            }
            break;
    }
    /* Return the information needed for updating the motor. */
    return(Return_Value);
}

/**
 * @brief This function sets the output voltage for the h-bridge.
 * @param Address The h-bridge I2C address.
 * @param Voltage_Value The equivalent voltage value for the h-bridge. Note
this is not the desired voltage.
 * @param Mode The direction for the h-bridge (0 Standby/coast, 1 Reverse, 2
Forward, 3 Brake).
 * @warning This function does not check if the mode is above 3 (the maximum
for the specified selection).
 */
inline void DRV8830_Set_Voltage(uint8_t Address, uint8_t Voltage_Value,
uint8_t Mode)
{
    /* Send the update voltage message to the h-bridge. */
    I2C_Interface_Write(Address, DRV8830_Control, (Voltage_Value<<2)|Mode);
}

/**
 * @brief This function sets the output voltage for the X-axis h-bridge. Note
that this function only Updates the h-bridge if the mode or voltage is
different.
 * @param Voltage The desired voltage for the motor to be at.
 */
void DRV8830_Set_Voltage_Motor_X(float Voltage)
{
    uint8_t Mode = DRV8830_Mode(&Voltage);
    /* Convert the voltage value to a h-bridge value.*/
    uint8_t Voltage_Value = DRV8830_Voltage_To_Int(Voltage);
    /* Check to if the voltage value and the mode are the same as the
previous values. */
    if (Voltage_Value_X_Old != Voltage_Value || Mode_X_Old != Mode)
    {
        /* Check the start up flag using the new values for the X motors. */
        Motor_X_Start_Up_Flag = DRV8830_Jump_Check(&Voltage_Value, &Mode,
Voltage_Value_X_Old, Mode_X_Old);
        /* Check the check the start up flags. */
        if(DRV8830_Start_Up_Flag_Control(Motor_Address_X))
        {
            /* Update the h-bridge voltage value for the X-axis motor. */
            DRV8830_Set_Voltage(Motor_Address_X, Voltage_Value, Mode);
            /* Update the previous voltage value for the X-axis motor. */
            Voltage_Value_X_Old = Voltage_Value;
            /* Update the previous mode for the X-axis motor. */
            Mode_X_Old = Mode;
        }
        #if H_Bridge_Print == 1
```

```
            /* Initialize a character array. */
            char Buffer[50];
            /* Setup the buffer to send. */
            sprintf(Buffer, "X V %u, M %u, F %u\r\n", Voltage_Value_X_Old,
Mode_X_Old, Motor_X_Start_Up_Flag);
            /* Send the string over USART. */
            USART_Send_String(Buffer);
        #endif
    }
}

/**
 * @brief This function sets the output voltage for the Y-axis h-bridge. Note
that this function only Updates the h-bridge if the mode or voltage is
different.
 * @param Voltage The desired voltage for the motor to be at.
 */
void DRV8830_Set_Voltage_Motor_Y(float Voltage)
{
    uint8_t Mode = DRV8830_Mode(&Voltage);
    /* Convert the voltage value to a h-bridge value.*/
    uint8_t Voltage_Value = DRV8830_Voltage_To_Int(Voltage);
    /* Check to if the voltage value and the mode are the same as the
previous values. */
    if (Voltage_Value_Y_Old != Voltage_Value || Mode_Y_Old != Mode)
    {
        /* Check the start up flag using the new values for the Y motors. */
        Motor_Y_Start_Up_Flag = DRV8830_Jump_Check(&Voltage_Value, &Mode,
Voltage_Value_Y_Old, Mode_Y_Old);
        /* Check the check the start up flags. */
        if(DRV8830_Start_Up_Flag_Control(Motor_Address_Y))
        {
            /* Update the h-bridge voltage value for the X-axis motor. */
            DRV8830_Set_Voltage(Motor_Address_Y, Voltage_Value, Mode);
            /* Update the previous voltage value for the X-axis motor. */
            Voltage_Value_Y_Old = Voltage_Value;
            /* Update the previous mode for the X-axis motor. */
           Mode_Y_Old = Mode;
        }
        #if H_Bridge_Print == 1
            /* Initialize a character array. */
            char Buffer[50];
            /* Setup the buffer to send. */
            sprintf(Buffer, "Y V %u, M %u, F %u\r\n", Voltage_Value_Y_Old,
Mode_Y_Old, Motor_Y_Start_Up_Flag);
            /* Send the string over USART. */
            USART_Send_String(Buffer);
        #endif
    }
}

/**
 * @brief This function sets the output voltage for the Z-axis h-bridge. Note
that this function only Updates the h-bridge if the mode or voltage is
different.
 * @param Voltage The desired voltage for the motor to be at.
 */
```

```
void DRV8830_Set_Voltage_Motor_Z(float Voltage)
{
    uint8_t Mode = DRV8830_Mode(&Voltage);
    /* Convert the voltage value to a h-bridge value.*/
    uint8_t Voltage_Value = DRV8830_Voltage_To_Int(Voltage);
    /* Check to if the voltage value and the mode are the same as the
previous values. */
    if (Voltage_Value_Z_Old != Voltage_Value || Mode_Z_Old != Mode)
    {
        /* Check the start up flag using the new values for the Z motors. */
        Motor_Z_Start_Up_Flag = DRV8830_Jump_Check(&Voltage_Value, &Mode,
Voltage_Value_Z_Old, Mode_Z_Old);
        /* Check the check the start up flags. */
        if(DRV8830_Start_Up_Flag_Control(Motor_Address_Z))
        {
            /* Update the h-bridge voltage value for the X-axis motor. */
            DRV8830_Set_Voltage(Motor_Address_Z, Voltage_Value, Mode);
            /* Update the previous voltage value for the X-axis motor. */
            Voltage_Value_Z_Old = Voltage_Value;
            /* Update the previous mode for the X-axis motor. */
            Mode_Z_Old = Mode;
        }
        #if H_Bridge_Print == 1
            /* Initialize a character array. */
            char Buffer[50];
            /* Setup the buffer to send. */
            sprintf(Buffer, "Z V %u, M %u, F %u\r\n", Voltage_Value_Z_Old,
Mode_Z_Old, Motor_Z_Start_Up_Flag);
            /* Send the string over USART. */
            USART_Send_String(Buffer);
        #endif
    }
}

/**
 * @brief This function sets the initial output voltage for all three h-
bridges.
 */
inline void DRV8830_Initialize(void)
{
    /* Initialize the X-axis motor to 0 volts. */
    DRV8830_Set_Voltage_Motor_X(0.0);
    /* Initialize the Y-axis motor to 0 volts. */
    DRV8830_Set_Voltage_Motor_Y(0.0);
    /* Initialize the Z-axis motor to 0 volts. */
    DRV8830_Set_Voltage_Motor_Z(0.0);
}

/**
 * @brief This function reads the fault byte for the h-bridge.
 * @param Address The h-bridge I2C address.
 * @return The current fault byte for the h-bridge.
 */
inline uint8_t DRV8830_Read_Fault(uint8_t Address)
{
    /* Read and send back the contents of the fault byte. */
    return(I2C_Interface_Single_Read(Address, DRV8830_Fault));
```

```
}

/**
 * @brief This function reads the fault byte for the X-axis h-bridge.
 * @return The current fault byte for the X-axis h-bridge.
 */
uint8_t DRV8830_Read_Fault_Motor_X(void)
{
    /* Return the fault byte for the X-axis motor. */
    return(DRV8830_Read_Fault(Motor_Address_X));
}

/**
 * @brief This function reads the fault byte for the Y-axis h-bridge.
 * @return The current fault byte for the Y-axis h-bridge.
 */
uint8_t DRV8830_Read_Fault_Motor_Y(void)
{
    /* Return the fault byte for the Y-axis motor. */
    return(DRV8830_Read_Fault(Motor_Address_Y));
}

/**
 * @brief This function reads the fault byte for the Z-axis h-bridge.
 * @return The current fault byte for the Z-axis h-bridge.
 */
uint8_t DRV8830_Read_Fault_Motor_Z(void)
{
    /** Return the fault byte for the Z-axis motor. */
    return(DRV8830_Read_Fault(Motor_Address_Z));
}

/**
 * @brief This function clears the fault byte for the h-bridge.
 * @param Address The h-bridge I2C address.
 */
inline void DRV8830_Clear_Fault(uint8_t Address)
{
    /* Send the I2C message to the h-bridge to clear the fault bit. */
    I2C_Interface_Write(Address, DRV8830_Fault, 1<<Clear_Fault_Bit);
}

/**
 * @brief This function clears the fault byte for the X-axis h-bridge.
 */
void DRV8830_Clear_Fault_Motor_X(void)
{
    /* Clear the fault byte for the X-axis motor. */
    DRV8830_Clear_Fault(Motor_Address_X);
}

/**
 * @brief This function clears the fault byte for the Y-axis h-bridge.
 */
void DRV8830_Clear_Fault_Motor_Y(void)
{
```

```
    /* Clear the fault byte for the Y-axis motor. */
    DRV8830_Clear_Fault(Motor_Address_Y);
}

/**
 * @brief This function clears the fault byte for the Z-axis h-bridge.
 */
void DRV8830_Clear_Fault_Motor_Z(void)
{
    /* Clear the fault byte for the Z-axis motor. */
    DRV8830_Clear_Fault(Motor_Address_Z);
}


void DRV8830_Reset_Motor_X(void)
{
    /* Clear fault previous fault values. */
    DRV8830_Clear_Fault_Motor_X();
    /* Set the motor voltage to 0. */
    DRV8830_Set_Voltage_Motor_X(0.0);
}

void DRV8830_Reset_Motor_Y(void)
{
    /* Clear fault previous fault values. */
    DRV8830_Clear_Fault_Motor_Y();
    /* Set the motor voltage to 0. */
    DRV8830_Set_Voltage_Motor_Y(0.0);
}

void DRV8830_Reset_Motor_Z(void)
{
    /* Clear fault previous fault values. */
    DRV8830_Clear_Fault_Motor_Z();
    /* Set the motor voltage to 0. */
    DRV8830_Set_Voltage_Motor_Z(0.0);
}

/**
 * @brief This function checks to see if the X-axis motor is failing.
 * Counting and resetting the motor faults to make sure the motor is able to
continue.
 * @return Whether or not the fault count is greater than a specified amount.
 */
uint8_t DRV8830_Check_Motor_X(void)
{
    /* Initialize the fault count for amount of motor failures. */
    static uint8_t Fault_Count_X = 0;
    /* Read the fault register from the X-axis motor h-bridge. */
    uint8_t Fault_Value = DRV8830_Read_Fault_Motor_X();
    if(Fault_Value&0x01)
    {
        /* Clear the fault register for the X-axis motor h-bridge. */
        DRV8830_Reset_Motor_X();
        /* Increment the fault count for the Y-axis motor. */
        Fault_Count_X++;
    }
```

```
    else if(Fault_Value)
    {
        /* Clear fault previous fault values. */
        DRV8830_Clear_Fault_Motor_X();
    }
    else
    {
        /* Clear the fault count for the X-axis motor. */
        Fault_Count_X = 0;
    }
    /* Return whether or not the fault count is greater than a specified
amount. */
    return(Fault_Count_X>Motor_Fault_Count_Max);
}

/**
 * @brief This function checks to see if the Y-axis motor is failing.
 * Counting and resetting the motor faults to make sure the motor is able to
continue.
 * @return Whether or not the fault count is greater than a specified amount.
 */
uint8_t DRV8830_Check_Motor_Y(void)
{
    /* Initialize the fault count for amount of motor failures. */
    static uint8_t Fault_Count_Y = 0;
    /* Read the fault register from the Y-axis motor h-bridge. */
    uint8_t Fault_Value = DRV8830_Read_Fault_Motor_Y();
    if(Fault_Value&0x01)
    {
        /* Clear the fault register for the Y-axis motor h-bridge. */
        DRV8830_Reset_Motor_Y();
        /* Increment the fault count for the Y-axis motor. */
        Fault_Count_Y++;
    }
    else if(Fault_Value)
    {
        /* Clear fault previous fault values. */
        DRV8830_Clear_Fault_Motor_Y();
    }
    else
    {
        /* Clear the fault count for the Y-axis motor. */
        Fault_Count_Y = 0;
    }
    /* Return whether or not the fault count is greater than a specified
amount. */
    return(Fault_Count_Y>Motor_Fault_Count_Max);
}

/**
 * @brief This function checks to see if the Z-axis motor is failing.
 * Counting and resetting the motor faults to make sure the motor is able to
continue.
 * @return Whether or not the fault count is greater than a specified amount.
 */
uint8_t DRV8830_Check_Motor_Z(void)
{
```

```
    /* Initialize the fault count for amount of motor failures. */
    static uint8_t Fault_Count_Z = 0;
    /* Read the fault register from the Z-axis motor h-bridge. */
    uint8_t Fault_Value = DRV8830_Read_Fault_Motor_Z();
    if(Fault_Value&0x01)
    {
        /* Clear the fault register for the Z-axis motor h-bridge. */
        DRV8830_Reset_Motor_Z();
        /* Increment the fault count for the Z-axis motor. */
        Fault_Count_Z++;
    }
    else if(Fault_Value)
    {
        /* Clear fault previous fault values. */
        DRV8830_Clear_Fault_Motor_Z();
    }
    else
    {
        /* Clear the fault count for the Z-axis motor. */
        Fault_Count_Z = 0;
    }
    /* Return whether or not the fault count is greater than a specified
amount. */
    return(Fault_Count_Z>Motor_Fault_Count_Max);
}

/**
 * @brief This function checks to see if any of the 3 motors is failing.
 * @return Whether or not the fault count is greater than a specified amount
for each motor.\n
 * Bit 0: Z-axis motor\n
 * Bit 1: Y-axis motor\n
 * Bit 2: X-axis motor\n
 */
uint8_t DRV8830_Check_Motors(void)
{
    /* Return the value for the fault for each of the motors. Simplifying the
return value into 3 bits.*/

return(DRV8830_Check_Motor_X()<<2|DRV8830_Check_Motor_Y()<<1|DRV8830_Check_Mo
tor_Z()<<0);
}
```

DRV8830.h

```
/**
 * @file DRV8830.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 3/20/2015\n Created: 11/8/2014 3:54:57 PM
 * @brief This file is code to read control the h-bridges.
 */

#ifndef DRV8830_H_
#define DRV8830_H_

/** @brief The maximum voltage that we will allow for the h-bridge. */
```

```
#define Motor_Max_Voltage 4.98

#include <stdint.h>

void DRV8830_Initialize(void);
void DRV8830_Set_Voltage_Motor_X(float Voltage);
void DRV8830_Set_Voltage_Motor_Y(float Voltage);
void DRV8830_Set_Voltage_Motor_Z(float Voltage);
uint8_t DRV8830_Read_Fault_Motor_X(void);
uint8_t DRV8830_Read_Fault_Motor_Y(void);
uint8_t DRV8830_Read_Fault_Motor_Z(void);
void DRV8830_Clear_Fault_Motor_X(void);
void DRV8830_Clear_Fault_Motor_Y(void);
void DRV8830_Clear_Fault_Motor_Z(void);
uint8_t DRV8830_Check_Motors(void);

#endif /* DRV8830_H_ */
```

LSM303.c

```
/**
 * @file LSM303.c
 * @author Nicholas Sikkema
 * @version Revision: 1.5
 * @date Last Updated: 1/22/2015\n Created: 10/30/2014 4:17:54 PM
 * @brief This file is code to control the LSM303.
 */

#define Device_DLH 0
#define Device_DLM 1
#define Device_DLHC 2
#define Device_D 3
#define Current_Device Device_D

#include <stdint.h>
#include <math.h>
#include "../../Microcontroller/Communication/I2C_Interface.h"
#include "LSM303.h"

enum Register_Address
{
    CTRL0             = 0x1F, // D
    CTRL1             = 0x20, // D
    CTRL_REG1_A       = 0x20, // DLH, DLM, DLHC
    CTRL2             = 0x21, // D
    CTRL_REG2_A       = 0x21, // DLH, DLM, DLHC
    CTRL_REG4_A       = 0x23, // DLH, DLM, DLHC
    CTRL5             = 0x24, // D
    CTRL6             = 0x25, // D
    CTRL7             = 0x26, // D
    OUT_X_L_A         = 0x28,

    CRA_REG_M         = 0x00, // DLH, DLM, DLHC
    CRB_REG_M         = 0x01, // DLH, DLM, DLHC
    MR_REG_M          = 0x02, // DLH, DLM, DLHC

    WHO_AM_I          = 0x0F, // D
```

```
    // dummy addresses for registers in different locations on different
devices;
    // the library translates these based on device type
    // value with sign flipped is used as index into translated_regs array

    OUT_X_H_M         = 1,
    OUT_X_L_M         = 2,
    OUT_Y_H_M         = 3,
    OUT_Y_L_M         = 4,
    OUT_Z_H_M         = 5,
    OUT_Z_L_M         = 6,
    // Update dummy_reg_count if registers are added here!

    // device-specific register addresses

    DLH_OUT_X_H_M     = 0x03,
    DLH_OUT_X_L_M     = 0x04,
    DLH_OUT_Y_H_M     = 0x05,
    DLH_OUT_Y_L_M     = 0x06,
    DLH_OUT_Z_H_M     = 0x07,
    DLH_OUT_Z_L_M     = 0x08,

    DLM_OUT_X_H_M     = 0x03,
    DLM_OUT_X_L_M     = 0x04,
    DLM_OUT_Z_H_M     = 0x05,
    DLM_OUT_Z_L_M     = 0x06,
    DLM_OUT_Y_H_M     = 0x07,
    DLM_OUT_Y_L_M     = 0x08,

    DLHC_OUT_X_H_M    = 0x03,
    DLHC_OUT_X_L_M    = 0x04,
    DLHC_OUT_Z_H_M    = 0x05,
    DLHC_OUT_Z_L_M    = 0x06,
    DLHC_OUT_Y_H_M    = 0x07,
    DLHC_OUT_Y_L_M    = 0x08,

    D_OUT_X_L_M       = 0x08,
    D_OUT_X_H_M       = 0x09,
    D_OUT_Y_L_M       = 0x0A,
    D_OUT_Y_H_M       = 0x0B,
    D_OUT_Z_L_M       = 0x0C,
    D_OUT_Z_H_M       = 0x0D
};

//Global Variables
static uint8_t Acc_Address = 0;
static uint8_t Mag_Address = 0;
#define Register_Count 6
static enum Register_Address Translated_Register[Register_Count+1];// index 0
not used
static float Mag_Min[3] = {-32767, -32767, -32767};
static float Mag_Max[3] = {32767, 32767, 32767};
static float Mag_From[3] = {0,0,0};
static int16_t Mag_Temp[3] = {0,0,0};
static int16_t Acc_Temp[3] = {0,0,0};
static float Forward_Heading = 0;
```

```
#define Heading_Stable_Value 2

#define TEST_REG_INVALID           -1
#define D_SA0_HIGH_ADDRESS         0x1D // D with SA0 high
#define D_SA0_LOW_ADDRESS          0x1E // D with SA0 low or non-D
magnetometer
#define NON_D_MAG_ADDRESS          0x1E // D with SA0 low or non-D
magnetometer
#define NON_D_ACC_SA0_LOW_ADDRESS  0x18 // non-D accelerometer with SA0 low
#define NON_D_ACC_SA0_HIGH_ADDRESS 0x19 // non-D accelerometer with SA0 high
#define Radians_To_Degrees_Conv    57.2957795131

#define X 0
#define Y 1
#define Z 2

#define D_WHO_ID    0x49
#define DLM_WHO_ID  0x3C

uint8_t LSM303_Test_Register(uint8_t Device_Address, uint8_t Register)
{
    uint8_t Message_Buff = I2C_Interface_Single_Read(Device_Address,
Register);
    if(I2C_Interface_Not_Busy())
    {
        return Message_Buff;
    }
    else
    {
        return TEST_REG_INVALID;
    }
}

void LSM303_Initialize()
{
    // determine Device type if necessary
    #if Current_Device == Device_D
        if(LSM303_Test_Register(D_SA0_HIGH_ADDRESS, WHO_AM_I) == D_WHO_ID)
        {
            Acc_Address = D_SA0_HIGH_ADDRESS;
            Mag_Address = D_SA0_HIGH_ADDRESS;
        }
        else
        {
            Acc_Address = D_SA0_LOW_ADDRESS;
            Mag_Address = D_SA0_LOW_ADDRESS;
        }

        // set device addresses and translated register addresses
        Translated_Register[OUT_X_L_M] = D_OUT_X_L_M;
        Translated_Register[OUT_X_H_M] = D_OUT_X_H_M;
        Translated_Register[OUT_Y_L_M] = D_OUT_Y_L_M;
        Translated_Register[OUT_Y_H_M] = D_OUT_Y_H_M;
        Translated_Register[OUT_Z_L_M] = D_OUT_Z_L_M;
        Translated_Register[OUT_Z_H_M] = D_OUT_Z_H_M;
```

```
        Mag_From[0] = 1;
        Mag_From[1] = 0;
        Mag_From[2] = 0;

        // Accelerometer

        // 0x57 = 0b01010111
        // AFS = 0 (+/- 2 g full scale)
        I2C_Interface_Write(Acc_Address, CTRL2, 0x00);

        // 0x57 = 0b01010111
        // AODR = 0101 (50 Hz ODR);AZEN = AYEN = AXEN = 1 (all axes enabled)
        I2C_Interface_Write(Acc_Address, CTRL1, 0x57);

        // Magnetometer

        // 0x64 = 0b01100100
        // M_RES = 11 (high resolution mode);M_ODR = 001 (6.25 Hz ODR)
        I2C_Interface_Write(Mag_Address, CTRL5, 0x64);

        // 0x20 = 0b00100000
        // MFS = 01 (+/- 4 gauss full scale)
        I2C_Interface_Write(Mag_Address, CTRL6, 0x20);

        // 0x00 = 0b00000000
        // MLP = 0 (low power mode off);MD = 00 (continuous-conversion mode)
        I2C_Interface_Write(Mag_Address, CTRL7, 0x00);
    #elif Current_Device == Device_DLHC
        // set device addresses and translated register addresses
        Acc_Address = NON_D_ACC_SA0_HIGH_ADDRESS;// DLHC doesn't have SA0 but
uses same acc address as DLH/DLM with SA0 high
        Mag_Address = NON_D_MAG_ADDRESS;
        Translated_Register[OUT_X_H_M] = DLHC_OUT_X_H_M;
        Translated_Register[OUT_X_L_M] = DLHC_OUT_X_L_M;
        Translated_Register[OUT_Y_H_M] = DLHC_OUT_Y_H_M;
        Translated_Register[OUT_Y_L_M] = DLHC_OUT_Y_L_M;
        Translated_Register[OUT_Z_H_M] = DLHC_OUT_Z_H_M;
        Translated_Register[OUT_Z_L_M] = DLHC_OUT_Z_L_M;

        Mag_From[0] = 0;
        Mag_From[1] = -1;
        Mag_From[2] = 0;

        // Accelerometer

        // 0x08 = 0b00001000
        // FS = 00 (+/- 2 g full scale);HR = 1 (high resolution enable)
        I2C_Interface_Write(Acc_Address, CTRL_REG4_A, 0x08);

        // 0x47 = 0b01000111
        // ODR = 0100 (50 Hz ODR);LPen = 0 (normal mode);Zen = Yen = Xen = 1
(all axes enabled)
        I2C_Interface_Write(Acc_Address, CTRL_REG1_A, 0x47);

        // Magnetometer
```

```
        // 0x0C = 0b00001100
        // DO = 011 (7.5 Hz ODR)
        I2C_Interface_Write(Mag_Address, CRA_REG_M, 0x0C);

        // 0x20 = 0b00100000
        // GN = 001 (+/- 1.3 gauss full scale)
        I2C_Interface_Write(Mag_Address, CRB_REG_M, 0x20);

        // 0x00 = 0b00000000
        // MD = 00 (continuous-conversion mode)
        I2C_Interface_Write(Mag_Address, MR_REG_M, 0x00);
    #elif Current_Device == Device_DLH
        // Device responds to address 0011000 with DLM ID;guess that it's a
DLM
        if(LSM303_Test_Register(NON_D_ACC_SA0_LOW_ADDRESS, CTRL_REG1_A) !=
TEST_REG_INVALID)
        {
            Acc_Address = NON_D_ACC_SA0_HIGH_ADDRESS;
        }
        else
        {
            Acc_Address = NON_D_ACC_SA0_LOW_ADDRESS;
        }

        // set device addresses and translated register addresses
        Mag_Address = NON_D_MAG_ADDRESS;
        Translated_Register[OUT_X_H_M] = DLH_OUT_X_H_M;
        Translated_Register[OUT_X_L_M] = DLH_OUT_X_L_M;
        Translated_Register[OUT_Y_H_M] = DLH_OUT_Y_H_M;
        Translated_Register[OUT_Y_L_M] = DLH_OUT_Y_L_M;
        Translated_Register[OUT_Z_H_M] = DLH_OUT_Z_H_M;
        Translated_Register[OUT_Z_L_M] = DLH_OUT_Z_L_M;

        Mag_From[0] = 0;
        Mag_From[1] = -1;
        Mag_From[2] = 0;

        // Accelerometer

        // 0x00 = 0b00000000
        // FS = 00 (+/- 2 g full scale)
        I2C_Interface_Write(Acc_Address, CTRL_REG4_A, 0x00);

        // 0x27 = 0b00100111
        // PM = 001 (normal mode);DR = 00 (50 Hz ODR);Zen = Yen = Xen = 1
(all axes enabled)
        I2C_Interface_Write(Acc_Address, CTRL_REG1_A, 0x27);

        // Magnetometer

        // 0x0C = 0b00001100
        // DO = 011 (7.5 Hz ODR)
        I2C_Interface_Write(Mag_Address, CRA_REG_M, 0x0C);

        // 0x20 = 0b00100000
        // GN = 001 (+/- 1.3 gauss full scale)
        I2C_Interface_Write(Mag_Address, CRB_REG_M, 0x20);
```

```
        // 0x00 = 0b00000000
        // MD = 00 (continuous-conversion mode)
        I2C_Interface_Write(Mag_Address, MR_REG_M, 0x00);
    #elif Current_Device == Device_DLM
        // set device addresses and translated register addresses
        Acc_Address = NON_D_ACC_SA0_HIGH_ADDRESS;
        Mag_Address = NON_D_MAG_ADDRESS;
        Translated_Register[OUT_X_H_M] = DLM_OUT_X_H_M;
        Translated_Register[OUT_X_L_M] = DLM_OUT_X_L_M;
        Translated_Register[OUT_Y_H_M] = DLM_OUT_Y_H_M;
        Translated_Register[OUT_Y_L_M] = DLM_OUT_Y_L_M;
        Translated_Register[OUT_Z_H_M] = DLM_OUT_Z_H_M;
        Translated_Register[OUT_Z_L_M] = DLM_OUT_Z_L_M;

        Mag_From[0] = 0;
        Mag_From[1] = -1;
        Mag_From[2] = 0;

        // Accelerometer

        // 0x00 = 0b00000000
        // FS = 00 (+/- 2 g full scale)
        I2C_Interface_Write(Acc_Address, CTRL_REG4_A, 0x00);

        // 0x27 = 0b00100111
        // PM = 001 (normal mode);DR = 00 (50 Hz ODR);Zen = Yen = Xen = 1
(all axes enabled)
        I2C_Interface_Write(Acc_Address, CTRL_REG1_A, 0x27);

        // Magnetometer

        // 0x0C = 0b00001100
        // DO = 011 (7.5 Hz ODR)
        I2C_Interface_Write(Mag_Address, CRA_REG_M, 0x0C);

        // 0x20 = 0b00100000
        // GN = 001 (+/- 1.3 gauss full scale)
        I2C_Interface_Write(Mag_Address, CRB_REG_M, 0x20);

        // 0x00 = 0b00000000
        // MD = 00 (continuous-conversion mode)
        I2C_Interface_Write(Mag_Address, MR_REG_M, 0x00);
    #endif
}


void LSM303_Read_Mag(int16_t *Mag)
{
    uint8_t Readings[6] = {0,0,0,0,0,0};
    #if Current_Device == Device_D

I2C_Interface_Read_Array(Mag_Address,Translated_Register[OUT_X_L_M]|(1<<7),Re
adings,6);
    #else
```

```
I2C_Interface_Read_Array(Mag_Address,Translated_Register[OUT_X_H_M],Readings,
6);
    #endif

    #if (Current_Device == Device_D)
        Mag[X] = (Readings[1]<<8|Readings[0]);
        Mag[Y] = (Readings[3]<<8|Readings[2]);
        Mag[Z] = (Readings[5]<<8|Readings[4]);
    #elif(Current_Device == Device_DLH)
        Mag[X] = (Readings[0]<<8|Readings[1]);
        Mag[Y] = (Readings[2]<<8|Readings[3]);
        Mag[Z] = (Readings[4]<<8|Readings[5]);
    #else
        Mag[X] = (Readings[0]<<8|Readings[1]);
        Mag[Y] = (Readings[4]<<8|Readings[5]);
        Mag[Z] = (Readings[2]<<8|Readings[3]);
    #endif
}


void LSM303_Read_Acc(int16_t *Acc)
{
    uint8_t Readings[6] = {0,0,0,0,0,0};
    I2C_Interface_Read_Array(Acc_Address,OUT_X_L_A,Readings,6);
    Acc[X] = (Readings[1]<<8|Readings[0]);
    Acc[Y] = (Readings[3]<<8|Readings[2]);
    Acc[Z] = (Readings[5]<<8|Readings[4]);
}

inline void LSM303_Array_Cross_Product(float *a, float *b, float *Output)
{
    Output[X] = (a[Y]*b[Z]) - (a[Z]*b[Y]);
    Output[Y] = (a[Z]*b[X]) - (a[X]*b[Z]);
    Output[Z] = (a[X]*b[Y]) - (a[Y]*b[X]);
}

inline float LSM303_Array_Dot_Product(float *a, float *b)
{
    return (a[X]*b[X]+a[Y]*b[Y]+a[Z]*b[Z]);
}

inline void LSM303_Array_Normalize(float *a)
{
    float Magnitude = sqrt(LSM303_Array_Dot_Product(a,a));
    a[X] /= Magnitude;
    a[Y] /= Magnitude;
    a[Z] /= Magnitude;
}

float LSM303_Heading(int16_t *Acc, int16_t *Mag)
{
    Mag[X] -= (Mag_Min[X]+Mag_Max[X])/2;
    Mag[Y] -= (Mag_Min[Y]+Mag_Max[Y])/2;
    Mag[Z] -= (Mag_Min[Z]+Mag_Max[Z])/2;
    float E[3] = {0,0,0};
    float N[3] = {0,0,0};
```

```
    float TempMag[3] = {Mag[X], Mag[Y], Mag[Z]};
    float TempAcc[3] = {Acc[X], Acc[Y], Acc[Z]};
    LSM303_Array_Cross_Product(TempMag,TempAcc,E);
    LSM303_Array_Normalize(E);
    LSM303_Array_Cross_Product(TempAcc,E,N);
    LSM303_Array_Normalize(N);
    float heading =
atan2(LSM303_Array_Dot_Product(E,Mag_From),LSM303_Array_Dot_Product(N,Mag_Fro
m))*Radians_To_Degrees_Conv;
    if(heading < 0)
    {
        heading += 360;
    }
    return heading;
}

void LSM303_UpdateForwardHeading(void)
{
    LSM303_Read_Mag(Mag_Temp);
    LSM303_Read_Acc(Acc_Temp);
    Forward_Heading = LSM303_Heading(Acc_Temp, Mag_Temp);
}

float LSM303_HeadingWithOffset(float heading)
{
    float Temp_Heading = heading - Forward_Heading;
    if(Temp_Heading < 0)
    {
        Temp_Heading += 360;
    }
    return Temp_Heading;
}

uint8_t LSM303_RotationFinished(void)
{
    LSM303_Read_Mag(Mag_Temp);
    LSM303_Read_Acc(Acc_Temp);
    float Heading = LSM303_HeadingWithOffset(LSM303_Heading(Acc_Temp,
Mag_Temp));
    static float Previous_Heading = 0;
    static uint8_t flag = 0;
    int Return_Value = 0;
    if((Heading<5&&Heading>-5)&&(Previous_Heading>5||Previous_Heading<-5))
    {
        if (flag == 0)
        {
            flag = 1;
        }
        else
        {
            Return_Value = 1;
        }
    }
    Previous_Heading = Heading;
    return(Return_Value);
}
```

```
uint8_t LSM303_IsStablized(void)
{
    static float Previous_Heading_Value = 1000;
    float Heading_Change = Forward_Heading-Previous_Heading_Value;
    Previous_Heading_Value = Forward_Heading;
    if(Heading_Change<Heading_Stable_Value||Heading_Change<-
Heading_Stable_Value)
    {
        return(1);
    }
    else
    {
        return(0);
    }
}
```

LSM303.h

```
/**
 * @file LSM303.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/22/2015\n Created: 10/30/2014 4:42:22 PM
 * @brief This file is code to control the LSM303.
 */

#ifndef LSM303_H_
#define LSM303_H_

#include <stdint.h>

void LSM303_Initialize();
void LSM303_Read_Mag(int16_t *Mag);
void LSM303_Read_Acc(int16_t *Acc);
float LSM303_Heading(int16_t *Acc, int16_t *Mag);
void LSM303_UpdateForwardHeading(void);
float LSM303_HeadingWithOffset(float heading);
uint8_t LSM303_RotationFinished(void);
uint8_t LSM303_IsStablized(void);

#endif /* LSM303_H_ */
```

CLS15.c

```
/**
 * @file CLS15.c
 * @author Nicholas Sikkema
 * @version Revision: 2.0
 * @date Last Updated: 1/27/2015\n Created: 12/2/2014 4:32:25 PM
 * @brief This file is code for the photodiode readings.
 */

#include <stdint.h>
#include "../../Microcontroller/ADC/ADC.h"
#include "../../Microcontroller/ADC/CD4051.h"
#include "CLS15.h"

/* Static variables to hold the photodiode values. */
```

```
static uint16_t Photodiode_Value_Front = 0;
static uint16_t Photodiode_Value_Right_Front = 0;
static uint16_t Photodiode_Value_Right_Rear = 0;
static uint16_t Photodiode_Value_Left_Front = 0;
static uint16_t Photodiode_Value_Left_Rear = 0;
/* Static variables to hold the photodiode offsets. */
static uint16_t Photodiode_Value_Front_Offset = 5;
static uint16_t Photodiode_Value_Right_Front_Offset = 5;
static uint16_t Photodiode_Value_Right_Rear_Offset = 5;
static uint16_t Photodiode_Value_Left_Front_Offset = 5;
static uint16_t Photodiode_Value_Left_Rear_Offset = 5;
/* Static variables to hold the ratios for the left and right sides. */
static float left_ratio = 0;
static float right_ratio = 0;

/**
 * @brief Determines which Photodiode is reading higher and creates a ratio
based on that knowledge.
 * @param Front_Value The current value for the front (left or right)
photodiode.
 * @param Rear_Value The current value for the rear (left or right)
photodiode.
 * @return The ratio between the front and rear photodiode values. If > 1
then it is in the front, otherwise it is in the rear.
 */
inline float CLS15_Ratio_Calculation(uint16_t Front_Value, uint16_t
Rear_Value)
{
    /** Return the ratio of the front value to the rear value. */
    return((float)Front_Value/(float)Rear_Value);
}

/**
 * @brief Update the front photodiode value.
 */
void CLS15_Update_Photodiode_Value_Front(void)
{
    Photodiode_Value_Front = CD4051_Read_Single_Pin(Photodiode_front)-
Photodiode_Value_Front_Offset;
}

/**
 * @brief Update the right photodiode values.
 */
void CLS15_Update_Photodiode_Value_Right(void)
{
    Photodiode_Value_Right_Front =
CD4051_Read_Single_Pin(Photodiode_rightfront)-
Photodiode_Value_Right_Front_Offset;
    Photodiode_Value_Right_Rear =
CD4051_Read_Single_Pin(Photodiode_rightrear)-
Photodiode_Value_Right_Rear_Offset;
    right_ratio = CLS15_Ratio_Calculation(Photodiode_Value_Right_Front,
Photodiode_Value_Right_Rear);
}

/**
```

```
 * @brief Update the left photodiode values.
 */
void CLS15_Update_Photodiode_Value_Left(void)
{
    Photodiode_Value_Left_Front =
CD4051_Read_Single_Pin(Photodiode_leftfront)-
Photodiode_Value_Left_Front_Offset;
    Photodiode_Value_Left_Rear = CD4051_Read_Single_Pin(Photodiode_leftrear)-
Photodiode_Value_Left_Rear_Offset;
    left_ratio = CLS15_Ratio_Calculation(Photodiode_Value_Left_Front,
Photodiode_Value_Left_Rear);
}

/**
 * @brief Update all of the photodiode values.
 */
void CLS15_Update_Photodiode_Values(uint8_t Current_LED)
{
    if(Current_LED == 1)
    {
        CLS15_Update_Photodiode_Value_Right();
        CLS15_Update_Photodiode_Value_Left();
    }
    else if(Current_LED == 2)
    {
        CLS15_Update_Photodiode_Value_Front();
        CLS15_Update_Photodiode_Value_Left();
    }
    else if(Current_LED == 4)
    {
        CLS15_Update_Photodiode_Value_Front();
        CLS15_Update_Photodiode_Value_Right();
    }
    else if(Current_LED == 8)
    {
        CLS15_Update_Photodiode_Value_Front();
        CLS15_Update_Photodiode_Value_Right();
        CLS15_Update_Photodiode_Value_Left();
    }
    else if (Current_LED == 16)
    {
        CLS15_Update_Photodiode_Value_Front();
    }
}

/**
 * @brief Retrieve the ADC value for the front photodiode.
 * @return The ADC value for the front photodiode.
 */
uint16_t CLS15_Retrieve_Photodiode_ADC_Front(void)
{
    return(Photodiode_Value_Front);
}

/**
 * @brief Retrieve the ADC value for the right front photodiode.
 * @return The ADC value for the right front photodiode.
```

```
 */
uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Front(void)
{
    return(Photodiode_Value_Right_Front);
}

/**
 * @brief Retrieve the ADC value for the right rear photodiode.
 * @return The ADC value for the right rear photodiode.
 */
uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Rear(void)
{
    return(Photodiode_Value_Right_Rear);
}

/**
 * @brief Retrieve the ADC value for the left front photodiode.
 * @return The ADC value for the left front photodiode.
 */
uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Front(void)
{
    return(Photodiode_Value_Left_Front);
}

/**
 * @brief Retrieve the ADC value for the left rear photodiode.
 * @return The ADC value for the left rear photodiode.
 */
uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Rear(void)
{
    return(Photodiode_Value_Left_Rear);
}

/**
 * @brief Retrieve the ratio for the right photodiodes.
 * @return The ratio for the right photodiodes.
 */
float CLS15_Retrieve_Photodiode_Ratio_Right(void)
{
    return(right_ratio);
}

/**
 * @brief Retrieve the ratio for the left photodiodes.
 * @return The ratio for the left photodiodes.
 */
float CLS15_Retrieve_Photodiode_Ratio_Left(void)
{
    return(left_ratio);
}

void CLS15_Update_Photodiode_Offset_Front(void)
{
    uint16_t Temp_Value = CD4051_Read_Single_Pin(Photodiode_front);
    if(Temp_Value<Photodiode_Value_Front_Offset)
    {
        Photodiode_Value_Front_Offset = Temp_Value;
```

```
    }
}

void CLS15_Update_Photodiode_Offset_Right_Front(void)
{
    uint16_t Temp_Value = CD4051_Read_Single_Pin(Photodiode_rightfront);
    if(Temp_Value<Photodiode_Value_Front_Offset)
    {
        Photodiode_Value_Right_Front_Offset = Temp_Value;
    }
}

void CLS15_Update_Photodiode_Offset_Right_Rear(void)
{
    uint16_t Temp_Value = CD4051_Read_Single_Pin(Photodiode_rightrear);
    if(Temp_Value<Photodiode_Value_Front_Offset)
    {
        Photodiode_Value_Right_Rear_Offset = Temp_Value;
    }
}

void CLS15_Update_Photodiode_Offset_Left_Front(void)
{
    uint16_t Temp_Value = CD4051_Read_Single_Pin(Photodiode_leftfront);
    if(Temp_Value<Photodiode_Value_Front_Offset)
    {
        Photodiode_Value_Left_Front_Offset = Temp_Value;
    }
}

void CLS15_Update_Photodiode_Offset_Left_Rear(void)
{
    uint16_t Temp_Value = CD4051_Read_Single_Pin(Photodiode_leftrear);
    if(Temp_Value<Photodiode_Value_Front_Offset)
    {
        Photodiode_Value_Left_Rear_Offset = Temp_Value;
    }
}

/**
 * @brief Update the photodiode offsets.
 */
void CLS15_Update_Photodiode_Offset(void)
{
    /* Update the front photodiode offset. */
    CLS15_Update_Photodiode_Offset_Front();
    /* Update the front photodiode offset. */
    CLS15_Update_Photodiode_Offset_Left_Front();
    /* Update the front photodiode offset. */
    CLS15_Update_Photodiode_Offset_Left_Rear();
    /* Update the front photodiode offset. */
    CLS15_Update_Photodiode_Offset_Right_Front();
}
```

CLS15.h

```
/**
 * @file CLS15.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/27/2015\n Created: 12/2/2014 4:32:25 PM
 * @brief This is the header file for the photodiode reading code.
 */

#ifndef CLS15_H_
#define CLS15_H_

void CLS15_Update_Photodiode_Values(uint8_t Current_LED);
uint16_t CLS15_Retrieve_Photodiode_ADC_Front(void);
uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Front(void);
uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Rear(void);
uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Front(void);
uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Rear(void);
uint16_t CLS15_Retrieve_Photodiode_ADC_Rear(void);
float CLS15_Retrieve_Photodiode_Ratio_Right(void);
float CLS15_Retrieve_Photodiode_Ratio_Left(void);
void CLS15_Update_Photodiode_Offset(void);

#endif /* CLS15_H_ */
```

XPEBBL.c

```
/**
 * @file XPEBBL.c
 * @author Nicholas Sikkema
 * @version Revision: 2.0
 * @date Last Updated: 1/22/2015\n Created: 12/2/2014 3:07:03 PM
 * @brief This file is code to control the 3 Watt LEDs.
 */

/** @brief Max number of states for the change LED function to be in. */
#define Cycle_State 3
/** @brief Bit position of the front LED */
#define Front_LED_Bit 5

#include <avr/io.h>
#include <stdint.h>
#include "XPEBBL.h"

/**
 * @brief Initialize the ports used for the LEDs.
 */
void XPEBBL_Initialize(void)
{
    /* Initialize the Led Ports to be an output. */
    DDRB |= (1<<DDB5)|(1<<DDB4)|(1<<DDB3);
}

/**
 * @brief Updates which led is on.
 * @param LED_Pin The pin to be set.
 * @warning This function does not check the input.
```

```
 */
void XPEBBL_Set_Pin(unsigned char LED_Pin)
{
    /* Clear the bits 3, 4, and 5. */
    PORTB &= 0xC7;
    /* Set the LED value in the Port. */
    PORTB |= LED_Pin;
}

/**
 * @brief This function toggles the front LED.
 */
uint8_t XPEBBL_Toggle_Front_LED()
{
    /* Initialize the previous value variable. */
    static uint8_t Previous_Value = 0;
    /* Initialize the current value variable. */
    uint8_t Current_Value = 0;
    /* Check if the previous value is zero. */
    if(Previous_Value == 0)
    {
        /* Enable the bit for the front LED. */
        Current_Value = 1 << Front_LED_Bit;
    }
    else
    {
        /* Clear the bit for the front LED. */
        Current_Value = 0;
    }
    /* Set the LED value. */
    XPEBBL_Set_Pin(Current_Value);
    /* Update the previous value. */
    Previous_Value = Current_Value;
    return(~(Current_Value>0));
}

/**
 * @brief Changes which LED that is currently on.
 */
uint8_t XPEBLL_Change_LED(void)
{
    /* Initialize the current pin variable. */
    static uint8_t Current_Pin = 0x08;
    /* Initialize the LED state variable. */
    static uint8_t LED_State = 0;
    /* Change LED State. */
    if(LED_State>Cycle_State)
    {
        /* Change the current LED that is on. */
        Current_Pin = Current_Pin << 1;
        /* Check to see if the current pin is in bounds. */
        if(Current_Pin > 0x20)
        {
            /* Reinitialize the current pin value. */
            Current_Pin = 0x08;
        }
        /* Reinitialize the state of the LED. */
```

```
        LED_State = 0;
        /* Set the LED value. */
        XPEBBL_Set_Pin(Current_Pin);
    }
    /* Turn off State. */
    else if(LED_State == Cycle_State)
    {
        /* Turn off the LEDs. */
        XPEBBL_Set_Pin(0x00);
    }
    /* Stay on State. */
    else
    {
        /* Make sure that the LED value is what is wanted. */
        XPEBBL_Set_Pin(Current_Pin);
    }
    /* Increment the state of the LEDs.*/
    LED_State++;
    /* Return the current pin. */
    return(Current_Pin>>3);
}
```

XPEBBL.h

```
/**
 * @file XPEBBL.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/22/2015\n Created: 12/2/2014 3:07:26 PM
 * @brief This file is code to control the 3 Watt LEDs.
 */

#ifndef XPEBBL_H_
#define XPEBBL_H_

void XPEBBL_Initialize(void);
void XPEBBL_Set_Pin(unsigned char LED_Pin);
uint8_t XPEBLL_Change_LED(void);
uint8_t XPEBBL_Toggle_Front_LED(void);

#endif /* XPEBBL_H_ */
```

Battery.c

```
/**
 * @file Battery.c
 * @author Nicholas Sikkema
 * @version Revision: 2.0
 * @date Last Updated: 1/22/2015\n Created: 11/13/2014 2:33:44 PM
 * @brief This file is code to read the battery voltage.
 */

/** @brief This is the lowest safe voltage value. */
#define Battery_Safe_Voltage 3.45

#include "../../Microcontroller/ADC/ADC.h"
#include "../../Microcontroller/ADC/CD4051.h"
#include "Battery.h"
```

```
/**
 * @brief This function is used to read the ADC port for the battery.
 * @return ADC value of the battery.
 */
inline int Battery_Read_ADC()
{
    /* Return the ADC value for the battery. */
    return(CD4051_Read_Single_Pin(Battery_Mux_Port));
}

/**
 * @brief This function is used to read the battery voltage.
 * @return Voltage of the battery.
 */
float Battery_Read_Voltage()
{
    /* Return the voltage value for the battery. */
    return(ADC_Convert_To_Voltage(Battery_Read_ADC()));
}

/**
 * @brief This function is used to check to see if the battery is below X
volts.
 * @return True (1) or False (0)
 */
int Battery_Check_Status()
{
    /* Return if the battery voltage is less than the safe voltage. */
    return(Battery_Read_Voltage()<Battery_Safe_Voltage);
}
```

Battery.h

```
/**
 * @file Battery.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 1/22/2015\n Created: 11/13/2014 2:34:28 PM
 * @brief This is the header file for the battery voltage reading code.
 */

#ifndef BATTERY_H_
#define BATTERY_H_

float Battery_Read_Voltage(void);
int Battery_Check_Status(void);

#endif /* BATTERY_H_ */
```

MPX5010GP.c

```
/**
 * @file MPX5010GP.c
 * @author Nicholas Sikkema
 * @version Revision: 3.0
 * @date Last Updated: 2/11/2015\n Created: 2/11/2014 11:50 PM
```

```
 * @brief This file is code to read the depth from the MPX5010GP pressure
sensor.
 */

/** @brief The full scale span for the conversion between Voltage and PSI. */
#define VFSS 4.7
/** @brief The offset voltage for the conversion between Voltage and PSI. */
#define V_Offset 0.12
/** @brief Max rated PSI for the pressure sensor. */
#define Max_PSI 1.45

#include "../../Microcontroller/ADC/ADC.h"
#include "../../Microcontroller/ADC/CD4051.h"
#include "MPX5010GP.h"

/**
 * @brief Converts the pressure sensor's ADC value to a depth in PSI.
 * @param ADC_Value The current ADC value for the pressure sensor.
 * @return The current pressure sensor depth in PSI.
 */
float MPX5010GP_Calculate_PSI(int ADC_Value)
{
    /* Convert the ADC value to a PSI value. */
    return((ADC_Convert_To_Voltage(ADC_Value)-V_Offset)*Max_PSI/VFSS);
}

/**
 * @brief Read the ADC with the correct multiplexer port selected
 * @return The current pressure sensor ADC value.
 */
inline int MPX5010GP_Read_Value()
{
    /* Retrieve the ADC value for the pressure sensor. */
    return(CD4051_Read_Single_Pin(Pressure_Sensor_Mux_Port));
}

/**
 * @brief Obtains the pressure sensor's depth in feet.
 * @return The current pressure sensor depth in feet.
 */
float MPX5010GP_Depth()
{
    /* Gather the PSI value and convert the value to a depth value. */

return(MPX5010GP_Calculate_PSI(MPX5010GP_Read_Value())*Foot_Water_Per_PSI);
}
```

MPX5010GP.h

```
/**
 * @file MPX5010GP.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 2/11/2015\n Created: 2/11/2014 11:50 PM
 * @brief This file is code to read the depth from the MPX5010GP pressure
sensor.
 */
```

```
#ifndef MPX5010GP_H_
#define MPX5010GP_H_

#ifndef Foot_Water_Per_PSI
    /** @brief The conversion factor between PSI and feet. */
    #define Foot_Water_Per_PSI 2.921
#endif

float MPX5010GP_Depth(void);
float MPX5010GP_Calculate_PSI(int ADC_Value);

#endif /* MPX5010GP_H_ */
```

Test_Battery.c

```
/**
 * @file Test_Battery.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:38:56 PM
 * @brief This file is for the testing of the battery code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Power/Battery.h"

/**
 * @brief This function is used to test the reading of the battery code.
 */
void Test_Battery(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    char Buffer_ADC_Value_Voltage[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[80] = {0x00};
    while(1)
    {
        float ADC_Value_Voltage = Battery_Read_Voltage();
        if(Battery_Check_Status() == 0)
        {
            USART_Send_String("Status: Good\r\n");
        }
        else
        {
```

```
            USART_Send_String("Status: Bad\r\n");
        }
        sprintf(Buffer,"%s V\r\n", USART_ftoa(ADC_Value_Voltage,
Buffer_ADC_Value_Voltage));
        USART_Send_String(Buffer);
        _delay_ms(500);
    }
}
```

Test_Battery.h

```
/**
 * @file Test_Multiplexer.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:39:10 PM
 * @brief This file is the header file for the testing of the battery code.
 */

#ifndef TEST_BATTERY_H_
#define TEST_BATTERY_H_

void Test_Battery(void);

#endif /* TEST_BATTERY_H_ */
```

Test_H_Bridge.c

```
/**
 * @file Test_H_Bridge.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015 \n Created: 2/28/2015 6:35:33 PM
 * @brief This file is for the testing of the I2C H-Bridge code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdint.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/H_Bridge/DRV8830.h"

void Test_H_Bridge_Motor_X_Increase(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
```

```
    {
        for(float i=-Motor_Max_Voltage; i<Motor_Max_Voltage; i+=0.07)
        {
            DRV8830_Set_Voltage_Motor_X(i);
            fault = DRV8830_Read_Fault_Motor_X();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}

void Test_H_Bridge_Motor_X_Decrease(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=Motor_Max_Voltage; i>-Motor_Max_Voltage; i-=0.07)
        {
            DRV8830_Set_Voltage_Motor_X(i);
            fault = DRV8830_Read_Fault_Motor_X();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}

void Test_H_Bridge_Motor_Y_Increase(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=-Motor_Max_Voltage; i<Motor_Max_Voltage; i+=0.07)
        {
            DRV8830_Set_Voltage_Motor_Y(i);
            fault = DRV8830_Read_Fault_Motor_Y();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}
```

```
void Test_H_Bridge_Motor_Y_Decrease(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=Motor_Max_Voltage; i>-Motor_Max_Voltage; i-=0.07)
        {
            DRV8830_Set_Voltage_Motor_Y(i);
            fault = DRV8830_Read_Fault_Motor_Y();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}

void Test_H_Bridge_Motor_Z_Increase(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=-Motor_Max_Voltage; i<Motor_Max_Voltage; i+=0.07)
        {
            DRV8830_Set_Voltage_Motor_Z(i);
            fault = DRV8830_Read_Fault_Motor_Z();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(1500);
        }
    }
}

void Test_H_Bridge_Motor_Z_Decrease(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=Motor_Max_Voltage; i>-Motor_Max_Voltage; i-=0.07)
        {
            DRV8830_Set_Voltage_Motor_Z(i);
```

```
            fault = DRV8830_Read_Fault_Motor_Z();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}

void Test_H_Bridge_Motor_ALL_Increase(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=-Motor_Max_Voltage; i<=Motor_Max_Voltage; i+=0.07)
        {
            DRV8830_Set_Voltage_Motor_X(i);
            fault = DRV8830_Read_Fault_Motor_X();
            DRV8830_Clear_Fault_Motor_X();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            DRV8830_Set_Voltage_Motor_Y(i);
            DRV8830_Clear_Fault_Motor_Y();
            fault = DRV8830_Read_Fault_Motor_Y();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            DRV8830_Set_Voltage_Motor_Z(i);
            DRV8830_Clear_Fault_Motor_Z();
            fault = DRV8830_Read_Fault_Motor_Z();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}


void Test_H_Bridge_Motor_ALL_Decrease(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    DRV8830_Initialize();
    uint8_t fault = 0;
    char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[50] = {0x00};
    while(1)
    {
        for(float i=Motor_Max_Voltage; i>=-Motor_Max_Voltage; i-=0.07)
```

```
        {
            DRV8830_Set_Voltage_Motor_X(i);
            fault = DRV8830_Read_Fault_Motor_X();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            DRV8830_Set_Voltage_Motor_Y(i);
            fault = DRV8830_Read_Fault_Motor_Y();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            DRV8830_Set_Voltage_Motor_Z(i);
            fault = DRV8830_Read_Fault_Motor_Z();
            sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i,
Buffer_i), fault);
            USART_Send_String(Buffer);
            _delay_ms(500);
        }
    }
}

/**
 * @brief This function is used to test the h-bridge code.
 */
void Test_H_Bridge(void)
{
    #if H_Bridge_Test_Mode == 0
        #if H_Bridge_Test_Direction == 0
            Test_H_Bridge_Motor_X_Increase();
        #elif H_Bridge_Test_Direction == 1
            Test_H_Bridge_Motor_X_Decrease();
        #endif
    #elif H_Bridge_Test_Mode == 1
        #if H_Bridge_Test_Direction == 0
            Test_H_Bridge_Motor_Y_Increase();
        #elif H_Bridge_Test_Direction == 1
            Test_H_Bridge_Motor_Y_Decrease();
        #endif
    #elif H_Bridge_Test_Mode == 2
        #if H_Bridge_Test_Direction == 0
            Test_H_Bridge_Motor_Z_Increase();
        #elif H_Bridge_Test_Direction == 1
            Test_H_Bridge_Motor_Z_Decrease();
        #endif
    #elif H_Bridge_Test_Mode == 3
        #if H_Bridge_Test_Direction == 0
            Test_H_Bridge_Motor_ALL_Increase();
        #elif H_Bridge_Test_Direction == 1
            Test_H_Bridge_Motor_ALL_Decrease();
        #endif
    #endif
}
```

Test_H_Bridge.h

```
/**
 * @file Test_H_Bridge.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:30:52 PM
 * @brief This file is the header file for the testing of the H-Bridge code.
 */

#ifndef TEST_H_BRIDGE_H_
#define TEST_H_BRIDGE_H_

int Test_H_Bridge(void);

#endif /* TEST_H_BRIDGE_H_ */
```

Test_I2C_Compass.c

```
/**
 * @file Test_I2C_Compass.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:35:33 PM
 * @brief This file is for the testing of the I2C Compass code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdint.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/IMU/LSM303.h"

void Test_I2C_Compass_Mag_Acc(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    LSM303_Initialize();
    int16_t Acc[3] = {0, 0, 0};
    int16_t Mag[3] = {0, 0, 0};
    char Buffer[50] = {0x00};
    while(1)
    {
        LSM303_Read_Mag(Mag);
        LSM303_Read_Acc(Acc);
        sprintf(Buffer,"Acc: %d, %d, %d Mag: %d, %d, %d\r\n", Mag[0], Mag[1],
Mag[2], Acc[0], Acc[1], Acc[2]);
        USART_Send_String(Buffer);
        _delay_ms(100);
    }
}
```

```
void Test_I2C_Compass_Heading(void)
{
    I2C_Interface_Initialize();
    USART_Initialize();
    LSM303_Initialize();
    int16_t Acc[3] = {0, 0, 0};
    int16_t Mag[3] = {0, 0, 0};
    char Buffer_Heading[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    float Heading = 0;
    char Buffer[10] = {0x00};
    while(1)
    {
        LSM303_Read_Mag(Mag);
        LSM303_Read_Acc(Acc);
        Heading = LSM303_Heading(Acc, Mag);
        sprintf(Buffer,"%s\r\n", USART_ftoa(Heading, Buffer_Heading));
        USART_Send_String(Buffer);
        _delay_ms(100);
    }
}

/**
 * @brief This function is used to test the compass code.
 */
void Test_I2C_Compass(void)
{
    #if Compass_Test_Mode == 0
        Test_I2C_Compass_Mag_Acc();
    #elif Compass_Test_Mode == 1
        Test_I2C_Compass_Heading();
    #endif
}
```

Test_I2C_Compass.h

```
/**
 * @file Test_I2C_Compass.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:35:13 PM
 * @brief This is the header file for the testing of the I2C Compass code.
 */

#ifndef TEST_I2C_COMPASS_H_
#define TEST_I2C_COMPASS_H_

void Test_I2C_Compass(void);

#endif /* TEST_I2C_COMPASS_H_ */
```

Test_LED.c

```
/**
 * @file Test_LED.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:31:18 PM
```

```
 * @brief This file is for the testing of the LED code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include "../Master_Build_Control.h"
#include "../Subsystem/Light/XPEBBL.h"

void Test_LED_Front_Toggle(void)
{
    /* Initialize the LEDs. */
    XPEBBL_Initialize();
    while(1)
    {
        XPEBBL_Toggle_Front_LED();
        _delay_ms(25);
    }
}

void Test_LED_Change_LED(void)
{
    /* Initialize the LEDs. */
    XPEBBL_Initialize();
    while(1)
    {
        XPEBLL_Change_LED();
        _delay_ms(25);
    }
}

/**
 * @brief This function is used to test the toggling of the LED code.
 */
void Test_LED(void)
{
    #if LED_Test_Mode == 0
        Test_LED_Front_Toggle();
    #elif LED_Test_Mode == 1
        Test_LED_Change_LED();
    #endif
}
```

Test_LED.h

```
/**
 * @file Test_LED.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:31:32 PM
 * @brief This is the header file for the testing of the LED code.
 */

#ifndef TEST_LED_H_
```

```
#define TEST_LED_H_

void Test_LED(void);

#endif /* TEST_LED_H_ */
```

Test_Motor_Control.c

```
/**
 * @file Test_Motor_Control.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:31:21 PM
 * @brief This file is for the testing of the motor control code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif
/**
 * @brief Update speeds for the motor control test function.\n
 * The rate is equal to (50 ms)*(variable value).
 * @warning This is value needs to be an integer.
 */
#define Motor_Control_Test_Speed 1000

#include <util/delay.h>
#include <stdio.h>
#include "../Algorithms/Motor_Control/Motor_Control.h"
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Subsystem/H_Bridge/DRV8830.h"
#include "../Subsystem/IMU/LSM303.h"

void Test_Motor_Control_X_Varying(void)
{
    I2C_Interface_Initialize();
    DRV8830_Initialize();
    USART_Initialize();
    CD4051_Initialize();
    Motor_Control_Motor_X_Flag_Update(1);
    char Buffer[50] = {0x00};
    while(1)
    {
        int Motor_Value = -5;
        for(int i=0; Motor_Value<=5; i++)
        {
            Motor_Control_Update_Velocity_X(Motor_Value);
            if(i%Motor_Control_Test_Speed==1)
            {
                Motor_Value++;
            }
            Motor_Control();
            sprintf(Buffer,"i: %d, Motor Value: %d\r\n", i, Motor_Value);
```

```
            USART_Send_String(Buffer);
            Motor_Control_Enable_Flag();
            _delay_ms(50);
        }
    }
}

void Test_Motor_Control_Y_Varying(void)
{
    I2C_Interface_Initialize();
    DRV8830_Initialize();
    USART_Initialize();
    CD4051_Initialize();
    Motor_Control_Motor_Y_Flag_Update(1);
    char Buffer[50] = {0x00};
    while(1)
    {
        int Motor_Value = -5;
        for(int i=0; Motor_Value<=5; i++)
        {
            Motor_Control_Update_Velocity_Y(Motor_Value);
            if(i%Motor_Control_Test_Speed==1)
            {
                Motor_Value++;
            }
            Motor_Control();
            sprintf(Buffer,"i: %d, Motor Value: %d\r\n", i, Motor_Value);
            USART_Send_String(Buffer);
            Motor_Control_Enable_Flag();
            _delay_ms(50);
        }
    }
}

void Test_Motor_Control_Z_Varying(void)
{
    I2C_Interface_Initialize();
    DRV8830_Initialize();
    USART_Initialize();
    CD4051_Initialize();
    Motor_Control_Motor_Z_Flag_Update(1);
    char Buffer[50] = {0x00};
    while(1)
    {
        int Motor_Value = 0;
        for(int i=0; Motor_Value<2.5; i++)
        {
            Motor_Control_Update_Depth_Z(Motor_Value);
            if(i%Motor_Control_Test_Speed==1)
            {
                Motor_Value++;
            }
            Motor_Control();
            sprintf(Buffer,"i: %d, Motor Value: %d\r\n", i, Motor_Value);
            USART_Send_String(Buffer);
            Motor_Control_Enable_Flag();
            _delay_ms(50);
```

```
        }
    }
}

void Test_Motor_Control_X_Constant(void)
{
    I2C_Interface_Initialize();
    DRV8830_Initialize();
    USART_Initialize();
    CD4051_Initialize();
    Motor_Control_Motor_X_Flag_Update(1);
    char Buffer[50] = {0x00};
    int Motor_Value = 0;
    while(1)
    {
        Motor_Control_Update_Velocity_X(Motor_Value);
        Motor_Control();
        sprintf(Buffer,"Motor Value: %d\r\n", Motor_Value);
        USART_Send_String(Buffer);
        Motor_Control_Enable_Flag();
        _delay_ms(50);
    }
}

void Test_Motor_Control_Y_Constant(void)
{
    I2C_Interface_Initialize();
    DRV8830_Initialize();
    USART_Initialize();
    CD4051_Initialize();
    Motor_Control_Motor_Y_Flag_Update(1);
    char Buffer[50] = {0x00};
    int Motor_Value = 0;
    while(1)
    {
        Motor_Control_Update_Velocity_Y(Motor_Value);
        Motor_Control();
        sprintf(Buffer,"Motor Value: %d\r\n", Motor_Value);
        USART_Send_String(Buffer);
        Motor_Control_Enable_Flag();
        _delay_ms(50);
    }
}

void Test_Motor_Control_Z_Constant(void)
{
    I2C_Interface_Initialize();
    DRV8830_Initialize();
    USART_Initialize();
    CD4051_Initialize();
    Motor_Control_Motor_Z_Flag_Update(1);
    char Buffer[50] = {0x00};
    int Motor_Value = 1;
    while(1)
    {
        Motor_Control_Update_Depth_Z(Motor_Value);
        Motor_Control();
```

```
        sprintf(Buffer,"Motor Value: %d\r\n", Motor_Value);
        USART_Send_String(Buffer);
        Motor_Control_Enable_Flag();
        _delay_ms(50);
    }
}

/**
 * @brief This function is used to test the motor control loops for the three
motors.
 */
void Test_Motor_Control(void)
{
    #if Motor_Control_Test_Mode == 0
        Test_Motor_Control_X_Varying();
    #elif Motor_Control_Test_Mode == 1
        Test_Motor_Control_Y_Varying();
    #elif Motor_Control_Test_Mode == 2
        Test_Motor_Control_Z_Varying();
    #elif Motor_Control_Test_Mode == 3
        Test_Motor_Control_X_Constant();
    #elif Motor_Control_Test_Mode == 4
        Test_Motor_Control_Y_Constant();
    #elif Motor_Control_Test_Mode == 5
        Test_Motor_Control_Z_Constant();
    #endif
}
```

Test_Motor_Control.h

```
/**
 * @file Test_Motor_Control.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:31:42 PM
 * @brief This file is the header file for the testing of the motor control
code.
 */

#ifndef TEST_MOTOR_CONTROL_H_
#define TEST_MOTOR_CONTROL_H_

void Test_Motor_Control(void);

#endif /* TEST_MOTOR_CONTROL_H_ */
```

Test_Multiplexer.c

```
/**
 * @file Test_Multiplexer.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:29:55 PM
 * @brief This file is for the testing of the multiplexer code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
```

```
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"

void Test_Multiplexer_Single_Pin(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    int ADC_Value_Pin = 0;
    char Buffer[20] = {0x00};
    while(1)
    {
        /* Read the ADC value. */
        ADC_Value_Pin = CD4051_Read_Single_Pin(0x0);
        /* Print the ADC value. */
        sprintf(Buffer,"%d\r\n", ADC_Value_Pin);
        USART_Send_String(Buffer);
        /* Delay between the last check. */
        _delay_ms(50);
    }
}

void Test_Multiplexer_Multiple_Pins(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    int ADC_Values[] = {0,0,0,0,0,0,0,0};
    char Buffer[100] = {0x00};
    while(1)
    {
        /* Read the range of ADC values. */
        CD4051_Read_Range(ADC_Values, 0x0, 0x7);
        /* Print the range of ADC values. */
        sprintf(Buffer,"%d, %d, %d, %d, %d, %d, %d, %d\r\n", ADC_Values[0],
ADC_Values[1], ADC_Values[2], ADC_Values[3], ADC_Values[4], ADC_Values[5],
ADC_Values[6], ADC_Values[7]);
        USART_Send_String(Buffer);
        /* Delay between the last check. */
        _delay_ms(50);
    }
}

void Test_Multiplexer_Voltage_Conversion()
{
    /* Initialize the multiplexer. */
```

```
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    int ADC_Value = 0;
    float Voltage_Value = 0;
    char Buffer_Voltage_Value[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[20] = {0x00};
    while(1)
    {
        /* Read the ADC value. */
        ADC_Value = CD4051_Read_Single_Pin(0x0);
        /* Convert the ADC value to a voltage. */
        Voltage_Value = ADC_Convert_To_Voltage(ADC_Value);
        /* Print the ADC and voltage value. */
        sprintf(Buffer,"%d, %s V\r\n", ADC_Value, USART_ftoa(Voltage_Value,
Buffer_Voltage_Value));
        USART_Send_String(Buffer);
        /* Delay between the last check. */
        _delay_ms(50);
    }
}

void Test_Multiplexer_Voltage_Multiple_Conversions()
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    int ADC_Value = 0;
    float Voltage_Value = 0;
    char Buffer_Voltage_Value[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[20] = {0x00};
    while(1)
    {
        /* Loop through the indexes for the multiplexer. */
        for(int i = 0; i < 7; i++)
        {
            /* Read the ADC value. */
            ADC_Value = CD4051_Read_Single_Pin(i);
            /* Convert the ADC value to a voltage. */
            Voltage_Value = ADC_Convert_To_Voltage(ADC_Value);
            /* Print the ADC and voltage value. */
            sprintf(Buffer,"%d, %s V\r\n", ADC_Value,
USART_ftoa(Voltage_Value, Buffer_Voltage_Value));
            USART_Send_String(Buffer);
        }
        /* Delay between the last check. */
        _delay_ms(50);
    }
}

/**
 * @brief This function is used to test the multiplexer ADC code.
 */
void Test_Multiplexer(void)
```

```
{
    #if Multiplexer_Test_Mode == 0
        Test_Multiplexer_Single_Pin();
    #elif Multiplexer_Test_Mode == 1
        Test_Multiplexer_Multiple_Pins();
    #elif Multiplexer_Test_Mode == 2
        Test_Multiplexer_Voltage_Conversion();
    #elif Multiplexer_Test_Mode == 3
        Test_Multiplexer_Voltage_Multiple_Conversions();
    #endif
}
```

Test_Multiplexer.h

```
/**
 * @file Test_Multiplexer.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:30:52 PM
 * @brief This file is the header file for the testing of the multiplexer
code.
 */

#ifndef TEST_MULTIPLEXER_H_
#define TEST_MULTIPLEXER_H_

void Test_Multiplexer(void);

#endif /* TEST_MULTIPLEXER_H_ */
```

Test_Photodiodes.c

```
/**
 * @file Test_Photodiodes.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:28:51 PM
 * @brief This file is for the testing of the photodiode code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Light/CLS15.h"
#include "../Subsystem/Light/XPEBBL.h"

void Test_Photodiodes_Print_Front(void)
{
    /* Initialize the temporary variables. */
    char Buffer[50] = {0x00};
```

```
    /* Retrieve the current photodiode ADC value. */
    uint16_t Photodiode_ADC = CLS15_Retrieve_Photodiode_ADC_Rear();
    /* Print the ADC information. */
    sprintf(Buffer,"Front: %u\r\n", Photodiode_ADC);
    USART_Send_String(Buffer);
}

void Test_Photodiodes_Print_Left(void)
{
    /* Initialize the temporary variables. */
    char Buffer[100] = {0x00};
    char Buffer_Photodiode_Ratio[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    /* Retrieve the current photodiode ADC values and ratio. */
    uint16_t Photodiode_ADC_Front =
CLS15_Retrieve_Photodiode_ADC_Left_Front();
    uint16_t Photodiode_ADC_Rear = CLS15_Retrieve_Photodiode_ADC_Left_Rear();
    float Photodiode_Ratio = CLS15_Retrieve_Photodiode_Ratio_Left();
    /* Print the ADC information. */
    sprintf(Buffer,"Left Front: %u Rear: %u Ratio: %s\r\n",
Photodiode_ADC_Front, Photodiode_ADC_Rear, USART_ftoa(Photodiode_Ratio,
Buffer_Photodiode_Ratio));
    USART_Send_String(Buffer);
}

void Test_Photodiodes_Print_Right(void)
{
    /* Initialize the temporary variables. */
    char Buffer[100] = {0x00};
    char Buffer_Photodiode_Ratio[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    /* Retrieve the current photodiode ADC values and ratio. */
    uint16_t Photodiode_ADC_Front =
CLS15_Retrieve_Photodiode_ADC_Right_Front();
    uint16_t Photodiode_ADC_Rear =
CLS15_Retrieve_Photodiode_ADC_Right_Rear();
    float Photodiode_Ratio = CLS15_Retrieve_Photodiode_Ratio_Right();
    /* Print the ADC information. */
    sprintf(Buffer,"Right Front: %u Rear: %u Ratio: %s\r\n",
Photodiode_ADC_Front, Photodiode_ADC_Rear, USART_ftoa(Photodiode_Ratio,
Buffer_Photodiode_Ratio));
    USART_Send_String(Buffer);
}

void Test_Photodiodes_Print_Rear(void)
{
    /* Initialize the temporary variables. */
    char Buffer[50] = {0x00};
    /* Retrieve the current photodiode ADC value. */
    int Photodiode_ADC = CLS15_Retrieve_Photodiode_ADC_Rear();
    /* Print the ADC information. */
    sprintf(Buffer,"Rear: %d\r\n", Photodiode_ADC);
    USART_Send_String(Buffer);
}

void Test_Photodiodes_Front(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
```

```
    /* Initialize USART communication. */
    USART_Initialize();
    while(1)
    {
        /* Update all of the photodiodes. */
        CLS15_Update_Photodiode_Values(8);
        /* Print the current value for the front photodiode. */
        Test_Photodiodes_Print_Front();
        /* Add a line between all of the reads. */
        USART_Send_String("\n");
        /* Delay between the last check. */
        _delay_ms(100);
    }
}

void Test_Photodiodes_Left(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    while(1)
    {
        /* Update all of the photodiodes. */
        CLS15_Update_Photodiode_Values(8);
        /* Print the current value for the left photodiode. */
        Test_Photodiodes_Print_Left();
        /* Add a line between all of the reads. */
        USART_Send_String("\n");
        /* Delay between the last check. */
        _delay_ms(100);
    }
}

void Test_Photodiodes_Right(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    while(1)
    {
        /* Update all of the photodiodes. */
        CLS15_Update_Photodiode_Values(8);
        /* Print the current value for the right photodiode. */
        Test_Photodiodes_Print_Right();
        /* Add a line between all of the reads. */
        USART_Send_String("\n");
        /* Delay between the last check. */
        _delay_ms(100);
    }
}

void Test_Photodiodes_Rear(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
```

```
    /* Initialize USART communication. */
    USART_Initialize();
    while(1)
    {
        /* Update all of the photodiodes. */
        CLS15_Update_Photodiode_Values(8);
        /* Print the current value for the rear photodiode. */
        Test_Photodiodes_Print_Rear();
        /* Add a line between all of the reads. */
        USART_Send_String("\n");
        /* Delay between the last check. */
        _delay_ms(100);
    }
}

void Test_Photodiodes_All(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    while(1)
    {
        /* Update all of the photodiodes. */
        CLS15_Update_Photodiode_Values(8);
        /* Print the current value for the front photodiode. */
        Test_Photodiodes_Print_Front();
        /* Print the current value for the left photodiode. */
        Test_Photodiodes_Print_Left();
        /* Print the current value for the right photodiode. */
        Test_Photodiodes_Print_Right();
        /* Print the current value for the rear photodiode. */
        Test_Photodiodes_Print_Rear();
        /* Add a line between all of the reads. */
        USART_Send_String("\n");
        /* Delay between the last check. */
        _delay_ms(100);
    }
}

void Test_Photodiodes_All_With_LED(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the LEDs. */
    XPEBBL_Initialize();
    while(1)
    {
        /* Change the current LED and all of the photodiodes that are not by
the current LED. */
        CLS15_Update_Photodiode_Values(XPEBLL_Change_LED());
        /* Print the current value for the front photodiode. */
        Test_Photodiodes_Print_Front();
        /* Print the current value for the left photodiode. */
```

```
        Test_Photodiodes_Print_Left();
        /* Print the current value for the right photodiode. */
        Test_Photodiodes_Print_Right();
        /* Print the current value for the rear photodiode. */
        Test_Photodiodes_Print_Rear();
        /* Add a line between all of the reads. */
        USART_Send_String("\n");
        /* Delay between the last check. */
        _delay_ms(200);
    }
}

/**
 * @brief This function is used to test the reading of the photodiodes code.
 */
void Test_Photodiodes(void)
{
    /* Test the front photodiode. */
    #if Photodiode_Test_Mode == 0
        Test_Photodiodes_Front();
    /* Test the left photodiodes. */
    #elif Photodiode_Test_Mode == 1
        Test_Photodiodes_Left();
    /* Test the right photodiode. */
    #elif Photodiode_Test_Mode == 2
        Test_Photodiodes_Right();
    /* Test the rear photodiode. */
    #elif Photodiode_Test_Mode == 3
        Test_Photodiodes_Rear();
    /* Test the all photodiodes. */
    #elif Photodiode_Test_Mode == 4
        Test_Photodiodes_All();
    /* Test the all photodiodes with LEDs. */
    #elif Photodiode_Test_Mode == 5
        Test_Photodiodes_All_With_LED();
    #endif
}
```

Test_Photodiodes.h

```
/**
 * @file Test_Photodiodes.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/11/2015\n Created: 2/28/2015 7:29:11 PM
 * @brief This is the header file for the testing of the photodiode code.
 */

#ifndef TEST_PHOTODIODES_H_
#define TEST_PHOTODIODES_H_

void Test_Photodiodes(void);

#endif /* TEST_PHOTODIODES_H_ */
```

Test_Pressure_Sensor.c

```
/**
 * @file Test_Pressure_Sensor.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:41:16 PM
 * @brief This file is for the testing of the pressure sensor code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Pressure/MPX5010GP.h"

/**
 * @brief This function is used to test the reading of the pressure sensor.
 */
void Test_Pressure_Sensor(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    int ADC_Value = 0;
    float ADC_Value_Voltage = 0;
    char Buffer_ADC_Value_Voltage[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    float PSI_Value = 0;
    char Buffer_PSI_Value[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    float Temp_Depth = 0;
    char Buffer_Temp_Depth[FLOATING_POINT_BUFFER_SIZE] = {0x00};
    char Buffer[80] = {0x00};
    while(1)
    {
        /* Read the adc pin value. */
        ADC_Value = CD4051_Read_Single_Pin(Pressure_Sensor_Mux_Port);
        /* Convert the adc value to a voltage. */
        ADC_Value_Voltage = ADC_Convert_To_Voltage(ADC_Value);
        /* Convert the voltage value to PSI. */
        PSI_Value = MPX5010GP_Calculate_PSI(ADC_Value);
        /* Convert the PSI to a depth. */
        Temp_Depth = PSI_Value*Foot_Water_Per_PSI;
        /* Print the pressure sensor information. */
        sprintf(Buffer,"%d, %s V, %s PSI, %s ft\r\n", ADC_Value,
USART_ftoa(ADC_Value_Voltage, Buffer_ADC_Value_Voltage),
USART_ftoa(PSI_Value, Buffer_PSI_Value), USART_ftoa(Temp_Depth,
Buffer_Temp_Depth));
        USART_Send_String(Buffer);
```

```
        /* Delay between the last check. */
        _delay_ms(50);
    }
}
```

Test_Pressure_Sensor.h

```
/**
 * @file Test_Pressure_Sensor.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:41:35 PM
 * @brief  This is the header file for the testing of the pressure sensor
code
 */

#ifndef TEST_PRESSURE_SENSOR_H_
#define TEST_PRESSURE_SENSOR_H_

void Test_Pressure_Sensor(void);

#endif /* TEST_PRESSURE_SENSOR_H_ */
```

Test_Swarming.c

```
/**
 * @file Test_Swarming.c
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 5:59:02 PM
 * @brief This file is for the testing of the swarming code.
 */

#ifndef F_CPU
    /** @brief The current clock speed for the microcontroller. */
    #define F_CPU 16000000UL
#endif

#include <util/delay.h>
#include <stdio.h>
#include "../Algorithms/Navigation/Swarming.h"
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Light/CLS15.h"

/**
 * @brief This function is used to test the swarming code.
 */
int Test_Swarming(void)
{
    /* Initialize the multiplexer. */
    CD4051_Initialize();
    /* Initialize USART communication. */
    USART_Initialize();
    /* Initialize the temporary variables. */
    char Buffer[50] = {0x00};
    while(1)
```

```
    {
        /* Update the photodiode values. */
        CLS15_Update_Photodiode_Values(8);
        /* Update the X and Y for the swarm algorithm. */
        Swarming();
        /* Obtain the X and Y from the swarm algorithm. */
        int X = Swarming_Retrieve_X_Offset();
        int Y = Swarming_Retrieve_Y_Offset();
        /* Print the swarm information. */
        sprintf(Buffer,"X: %d, Y: %d\r\n", X, Y);
        USART_Send_String(Buffer);
        /* Delay between the last check. */
        _delay_ms(100);
    }
}
```

Test_Swarming.h

```
/**
 * @file Test_Swarming.h
 * @author Nicholas Sikkema
 * @version Revision: 1.0
 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 5:59:18 PM
 * @brief This is the header file for the testing of the swarming code.
 */

#ifndef TEST_SWARMING_H_
#define TEST_SWARMING_H_

void Test_Swarming(void);

#endif /* TEST_SWARMING_H_ */
```

Code Documentation

To view the Doxygen documentation, navigate to: http://cegt201.bradley.edu/projects/proj2015/autonomous_underwater_robots/Deliverables/Doxygen_Report_4_30_2015/index.xhtml.

(I) Datasheets

To view the sources used download our project repo at: http://cegt201.bradley.edu/projects/proj2015/autonomous_underwater_robots/Deliverables/aquatic-robots-code-repo.zip.

(J) Detailed Distribution of Tasks

TABLE IV
Detailed Distribution of Tasks

Task Name	Resource Names
Research - Sensors - Sub Detection	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Research - Surfacing Techniques	Cameron Putz, Ryan Lipski
Research and Test - Sealing/Pressure	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Research - Power System	Cameron Putz, Ryan Lipski
Research - Sensors and Algorithms - Accelerometer	Nicholas Sikkema
Research - Sensors and Algorithms - Compass	Nicholas Sikkema
Design - Mock Drawings	Ryan Lipski
Research - PCB Construction Method	Cameron Putz
Research - Sensors and Algorithms - Bottom Detection	Nicholas Sikkema, Ryan Lipski
Research - Submarines	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Proposal Presentation	
Design - Amplifier Circuit	Cameron Putz
Research - Sensors and Algorithms - Pressure	Nicholas Sikkema
Research - Sensor Algorithms – LED Detection	Cameron Putz
Proposal Paper	
Design - Motor Control	Ryan Lipski
Research - Microcontroller	Nicholas Sikkema, Cameron Putz, Ryan Lipski
Design - Software Structure	Nicholas Sikkema
Research - Image Capturing	Nicholas Sikkema
Design - Algorithms - Detection Array	Cameron Putz
Design – Algorithms - Compass	Nicholas Sikkema
Design - Algorithms - Motor Control	Ryan Lipski
Testing/Tuning - Bench Top - Detection Array	Cameron Putz
Webpage Release	
Design – Algorithms - Pressure	Nicholas Sikkema, Ryan Lipski
Design - Algorithms - Accelerometer	Nicholas Sikkema, Ryan Lipski
Design – Algorithms – Bottom Detection	Nicholas Sikkema, Ryan Lipski
Design – Power System	Cameron Putz, Ryan Lipski
Design - PCB Circuit Layout	Cameron Putz
Design - Submarine Layout	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Design – Position Guidance Algorithm	Nicholas Sikkema, Ryan Lipski
Assemble - Single Submarine	Cameron Putz
Testing/Tuning - Single Submarine	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Fall Progress Presentation	
Research and Test - Photo Stitching Software	Nicholas Sikkema
Testing/Tuning - Bench Top - Directional Guidance	Nicholas Sikkema, Ryan Lipski
Performance Review	

Research - Swarming Techniques	Nicholas Sikkema, Ryan Lipski, Cameron Putz
Assembly - Swarm	Ryan Lipski, Cameron Putz
Simulate - Swarm Algorithm	Nicholas Sikkema, Ryan Lipski, Cameron Putz
Design - Swarm Algorithm	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Testing/Tuning - Bench Top - Swarm	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Spring Progress Presentation	
Testing/Tuning - Swarm	Cameron Putz, Nicholas Sikkema, Ryan Lipski
Project Demonstration	
Final Presentation	
Report Draft	
Student Expo	
ECE Advisory Board Poster Presentation	
Final Report	

(K) Recommendations for Future Work

One of the things the team recommends for future work is a wireless programmer. A wireless programmer would allow the user to debug the software in water significantly faster. The team had to remove the hot glue seal, pry open the sail, and then apply hot glue to the sail every time software was changed. The sealing process took around 60 minutes to complete. This could be have been avoided with a wireless programmer.

Instead of designing a custom sail, the AUR team would recommend purchasing a waterproof container to hold all of the electronics. The sail used by the team was very cost-efficient and able to cut to an exact size, but it was difficult to waterproof. Purchasing a container that was waterproof from the start would eliminate much of the waterproofing challenge.

Building the entire submarine platform from scratch could also be advantageous. The team decided to purchase an existing platform, as it was relatively inexpensive and reduced the amount of construction work required. An entirely custom submarine platform would allow the user to have more room in a larger sail. The user could select their own motors to create a more agile platform than the one the team purchased.