DEVELOPMENT OF ELECTRONICS, SOFTWARE, AND GRAPHICAL USER CONTROL INTERFACE FOR A WALL-CLIMBING ROBOT

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Lynda Beatriz Tesillo

June 2015

COMMITTEE MEMBERSHIP

TITLE:	Development of Electronics, Software, and Graphical User Control Interface for a Wall-Climbing Robot
AUTHOR:	Lynda Beatriz Tesillo
DATE SUBMITTED:	June 2015
COMMITTEE CHAIR:	Thomas Mackin, Ph.D. Professor of Mechanical Engineering
COMMITTEE MEMBER:	Xi (Julia) Wu, Ph.D. Associate Professor of Mechanical Engineering
COMMITTEE MEMBER:	John Ridgely, Ph.D. Professor of Mechanical Engineering

ABSTRACT

Development of Electronics, Software, and Graphical User

Control Interface for a Wall-Climbing Robot

Lynda Beatriz Tesillo

The objective for this project is to investigate various electrical and software means of control to support and advance the development of a novel vacuum adhesion system for a wall-climbing robot. The design and implementation of custom electronics and a wirelessly controlled real-time software system used to define and support the functionalities of these electronics is discussed. The testing and evaluation of the overall system performance and the performance of the several different subsystems developed, while working both individually and cooperatively within the system, is also demonstrated.

Keywords: Robot electronics, wall-climbing, vacuum adhesion, sliding tread adhesion, graphical user interface, custom circuit board, robot control

ACKNOWLEDGMENTS

I would first like to thank Dr. Thomas Mackin for having faith in me to successfully complete this project and providing me with this amazing learning opportunity. I would also like to thank Dr. John Ridgely for always being willing to answer my numerous spontaneous questions and sharing with me his vast knowledge in electronics and software. I also thank Dr. Xi (Julia) Wu for her dedication to my educational development and her constant enthusiasm and happiness that always put a smile on my face. I could not have asked for a better, more skilled committee to guide and accompany me through this project and I hope to be able to work with them all again in the future. I would also like to thank Jim Stefani for diligently working alongside me on this project and always being there for me when I needed help. His constant support and knowledgeable insight have proven to me that he will be successful in all he does in the future and I hope to continue the great friendship we have developed for many years to come. Many thanks to my mother as well, who constantly supported me and encouraged me to always work hard to be the best I could be. Lastly, many thanks to Brian Austin for always being there for me and providing me with unending encouragement and support in all of my successes and failures. Without all of these wonderful people playing such important roles in my educational development, I would not be where I am today. Many thanks again to all of you!

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Objective

The desire to develop autonomous robotic systems has driven significant advancements in electromechanical and computer technology over the past few decades. Thanks to these advancements, robots of all shapes and sizes have been, and are being, developed to engage in a vast array of tasks and functions. The majority of these robots are ground-based, mimicking terrestrial locomotors. The past few years have witnessed explosive growth in unmanned aerial vehicles that extend aircraft operating principles to autonomous control. There are several robots designed to mimic birds or insects (e.g. Harvard's RoboBee[1] and Aerovironments Hummingbird[2]), but also extend the domain of known aerodynamic vehicles to robotic systems. More recently, there is growing interest in robots that can operate on vertical or inverted surfaces. Robotic wall-climbing capabilities could be revolutionary in numerous applications, including surveillance, maintenance, and inspection, especially in dangerous environments[3][4][5]. Many semi-autonomous and autonomous wall-climbing robots have been prototyped, but most of these robots are still far from fully developed and their functionalities remain limited, especially with regards to payload, battery life, and reliability of the adhesion system during operation. There has not yet been a stand-out solution, encouraging others to ideate other solutions for the adhesion method and electronics and software that support that method's execution. The objective for this project is to explore various electrical and software means of control to support and advance a novel adhesion system for wall-climbing robots. In this domain, the design and implementation of a wireless control system, including testing and evaluation of system performance, is demonstrated.

Project Background

Previous work by von Broekhoven, Stefani, and Mackin[6] at California Polytechnic State University, San Luis Obispo demonstrated the feasibility of a novel adhesion method that utilized perforated RC tank treads as the adhesion interface for a wall-climbing robot. Their selection of a nominally smooth vertical surface proved crucial in defining the compliance and structure of the treads for their prototype adhesion and drive system. Furthermore, they chose a tethered system that supplied both power and compressed air to their machine, eliminating the need for heavy onboard power supplies and air pumps. Their adhesion system utilized a set of Venturi tubes that generated vacuum suction across the driving treads. This vacuum suction was generated by passing compressed air through an air distribution manifold to an array of Venturi tubes fixed to another manifold that supported the treads. This manifold was machined with an array of holes that were carefully chosen to interact with a hole pattern in the tread to create and distribute a uniform adhesion force across the tread and to the wall. The tread essentially behaved as a gasket, forming a tight seal between the manifold and the wall and creating several small spaces through which the vacuum pressure could adhere the robot to the wall. The initial prototype of this wall-climbing robot can be seen in Figure 1. After performing proof-of-concept testing for this tread-gasket vacuum adhesion method, the development of the robot hardware, software, and electronics could begin. At that point, preliminary mechanical hardware was designed and prototyped, but only a pair of actuating DC motors had been selected and no electronics or software had been developed. The original motors selected were not able to provide enough torque to drive the robot while adhered to the wall. Therefore, new actuators needed to be selected and implemented as well. The final mechanical design model selected can be seen in Figure 2.

Figure 1: Wall-climbing robot prototype adhered to a vertical surface that demonstrated proof-of-concept of a tread-gasket vacuum adhesion method

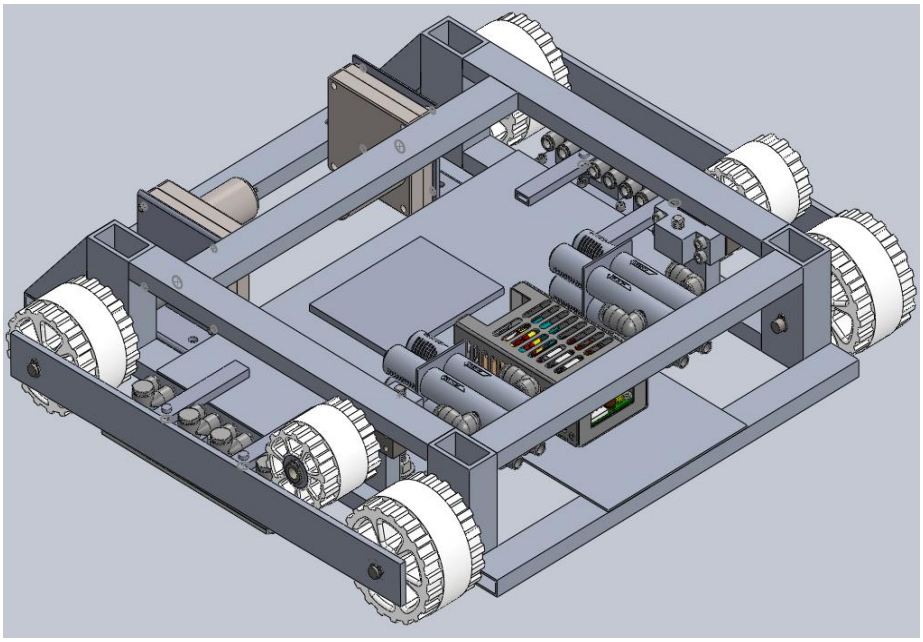

Figure 2: Final wall-climbing robot design used as the basis for the development of the electronics, software, and user control interface

Project Specifications

In order to investigate successful solutions, it was necessary to identify the desired attributes and functionalities that characterize successful wall-climbing robot electronics and software. First, the robot must be able to move in all directions (forward, reverse, left, and right), thus capable of linear and angular movement. The robot's speed and trajectory must be wirelessly controlled through a graphical user interface, which should also accurately display the current speed and position. Since vacuum suction was the chosen method of adhesion, it is important to have a system in place to monitor this adhesion during the robot's operation. In order to do this, the pressure in the air supply lines must be monitored and displayed in real-time through the graphical user interface. This ensures that the system will behave in a predictable manner and allow the user to respond to any potential loss of suction that may result in the robot losing contact with the wall. Additionally, the system must incorporate a simple means of crash and obstacle avoidance to prevent the robot from being damaged during operation. The weight of the electronic components must also be considered. Since the robot will have a set payload capacity, it is important to avoid taking up some of that capacity due to the utilization of overly sized and heavy electronics. Also, the amount of power that the electronics consume must be minimized. Low power consumption equates to a smaller and lighter power supply, taking up less payload capacity, and less heat generated by the electronics, increasing the potential continuous run-time of the robot system before overheating. The electronics and software must also be as simple as possible while still accomplishing the required tasks, as simplicity of the system allows for easier troubleshooting during the development process. Lastly, the system must be adaptable, meaning that additional electronics and software can later be added with ease, if necessary.

II. BACKGROUND RESEARCH

Existing Solutions

Numerous wall-climbing robots have already been developed, so it is important to identify the electrical and software-based technologies utilized in these robots and analyze their performance and success. There are several kinds of wall-climbing robots that have already been made. They are typically categorized based on their chosen method of adhesion, with the main methods being suction, magnetism, and grasping. Within these categories, what sets one robot apart from the rest are the electronics and software that are used to accomplish the robot's control, obtain sensory feedback, and interpret and process the acquired information in the appropriate fashion. Additionally, the overall size of the robot can also be a strong indicator of the success of the robot, especially because the application for which the robot is designed could have associated size restrictions or limitations.

Mahajan et al.[7] demonstrated a climbing robot for cleaning applications that used custom circuitry and onboard power for data acquisition, drive, and control. A microcontroller powered by a 6 volt lithium battery pack and connected to several DC motors and a pressure sensor controlled the functionality of the robot. The motors provided linear motion to the robot, while the pressure sensor monitored the vacuum pressure inside the suction cups used for wall adhesion. Overall, this robot did not appear to have strong motion or system monitoring and control capabilities and was described as being oversized, but was able to benefit from the flexible functionality that comes with using custom circuitry powered by a selected microcontroller instead of a mass manufactured data acquisition board.

While some robots are controlled by custom circuitry using carefully selected microcontrollers, other robots, such as the four-legged slider-crank suction powered robot by Albagul et al.[8], employed pre-made microcontroller modules. With these systems, all of the supporting circuitry has already been developed and integrated with the microcontroller, allowing the user to simply plug in their equipment and easily integrate electronic control into their system. This particular robot, pictured in Figure 3, utilized the BASIC Stamp BS2 module, which runs using the PBASIC programming language through a serial connection to a computer. A program was created and partitioned to control the various functionalities of the robot, including forward and reverse motion. The program was coded to operate very systematically and repeatedly perform the same tasks associated with forward and reverse motion while monitoring the state of four pneumatic valves to determine when the appropriate motors were powered to activate the slider-crank walking mechanisms. The flowchart of the program can be seen in Figure 4. This robot had several mechanical design issues related to the performance of the suction cups, but the ready-to-use electronics and task-oriented software were kept relatively simple and were overall successful. The only issue that was found was with regards to the power supply for the system. Because there were several motors, pneumatic valves, and a circuit board being actively controlled, a steady power supply was imperative, but this robot only relied on regular 9 volt batteries. This caused issues because of the fast discharge rate and cost associated with these batteries, which made powering the robot electronics very unreliable and expensive. The electronics were all housed onboard, so no tethering was required, preventing mobility issues associated with caring for tethered lines.

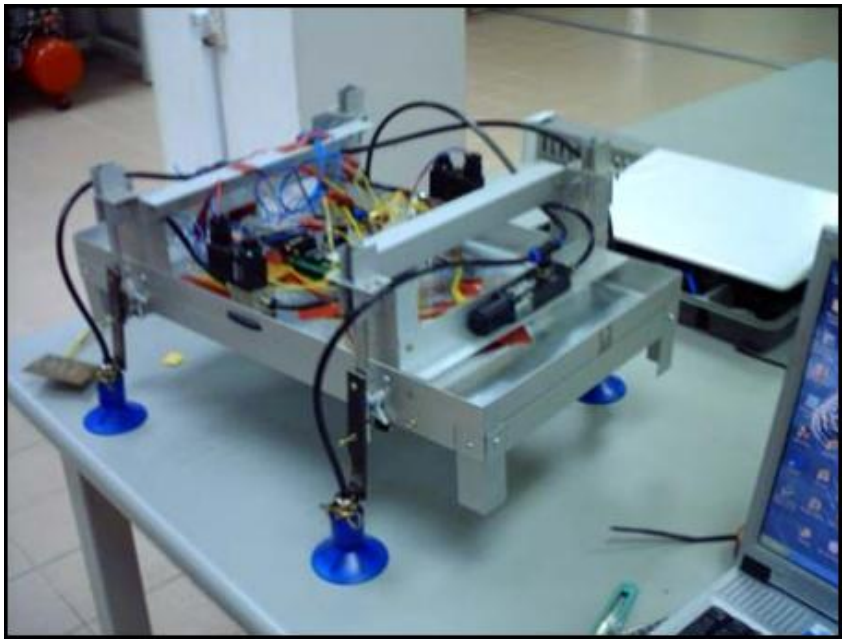

Figure 3: Four-legged slider-crank suction powered wall-climbing robot system

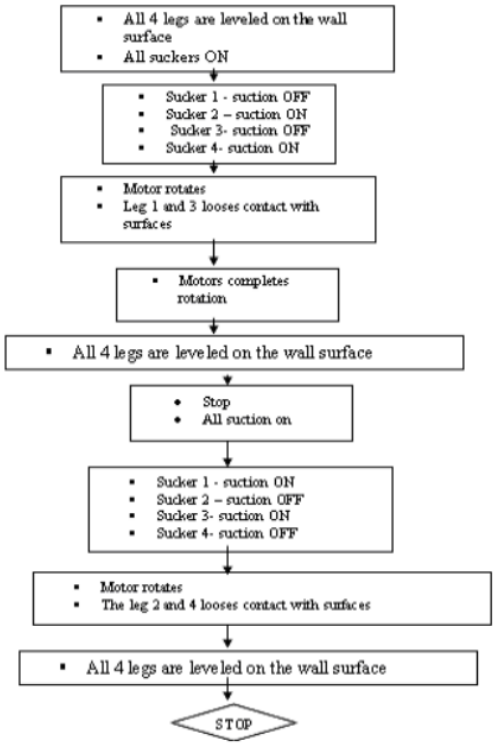

Figure 4: PBASIC programming flowchart for slider-crank wall-climbing robot

Another project that implements pre-manufactured microcontroller modules is the project aimed towards developing several design solutions for a tree-climbing robot[9]. Although the robots ideated for this project are not climbing walls, they still attempt to scale a vertical surface, in this case, a tree, using a multitude of different adhesion methods, including motor-actuated spikes. This project used Arduino microcontroller modules. Arduino boards are made for the everyday electronics hobbyist, making them very easy to use and implement into any simple system. There is also a very large database that provides Arduino users with a vast variety of code to run on the microcontrollers, so building up the code for any project is much easier than starting from scratch with another separately selected microcontroller. Arduino systems are also very inexpensive and modular, as they can be easily built upon in order to increase the functionality of the board. One such tree-climbing robots, along with the Arduino microcontroller module that was used to control it, can be seen in Figure 5. The report for this project also goes into great detail regarding the software created for the robots. These tree-climbing robots used serial communication to transfer data between the Arduino board on the robot and the user's computer configured with a graphical user interface prototype. The code that ran on the microcontroller controlled the robot using a decision-based methodology, meaning that the sensory inputs from the robot and the user inputs, commands for what the user wants the robot to do next, would be used to make decisions regarding the next actions of the robot and then immediately implement those decisions. A flowchart of this software architecture is presented in Figure 6. The graphical user interface was only partially developed using Qt Creator, so the development of manual control of the robot was not completed. However, the interface was kept simple in order to make the overall user experience an easy one.

Figure 5: One of several developed ideas for a tree-climbing robot using an Arduino microcontroller module for control and data acquisition

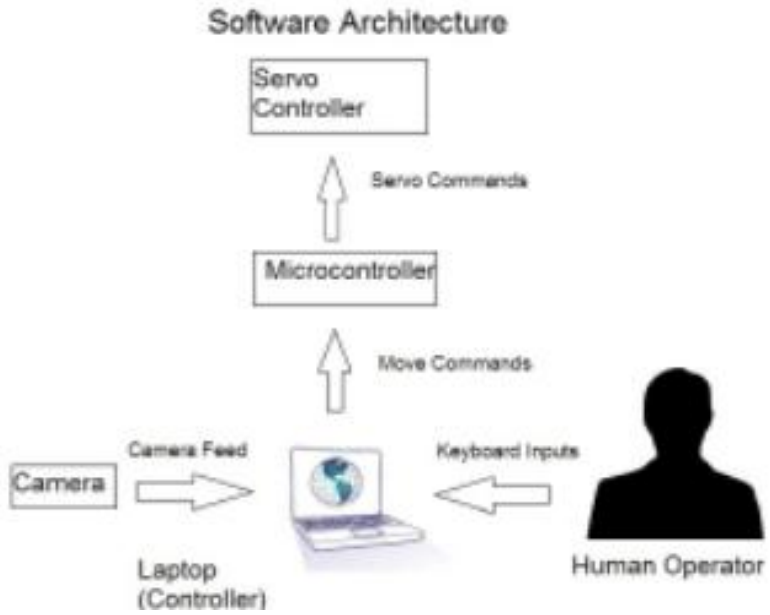

Figure 6: Overview of software architecture used for tree-climbing robot designs

Many robots have developed ingenious adhesion systems that actively maintain contact between the base of the robot and the wall. One team ideated a robot, seen in Figure 7, whose drive belts function to not only drive the robot forward or backwards, but also to engage and disengage the suction to the wall[10]. This was accomplished by securing 24 suction cups perpendicular to the drive belts, which are cyclically pressed against and pulled off of the wall, creating suction at each active suction cup. A DC motor, in this case, served a dual purpose, in that it dictated the robot's motion and activated the adhesion system. This reduced the overall weight and number of components required for the drive system and the adhesion system. If the drive system had not been integrated with the adhesion method, the robot most likely would have had a larger footprint. Most wall-climbing robots that have been developed incorporate motors of some kind to create the motion for the robot, but this particular robot was able to utilize it to serve a unique, additional function. This robot also is controlled using a custom programmable logic controller powered by a rechargeable battery. It also achieves wireless control using a radio controller. Another very similar robot[11] executes wireless control using a Bluetooth protocol. All in all, this robot successfully executes wireless control and takes advantage of the flexibility associated with using a custom circuit design.

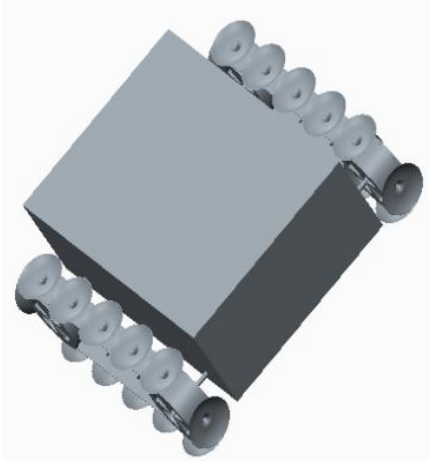

Figure 7: Robot developed with suction cups mounted to drive belts used for adhesion

Another example of assigning dual purpose to an electronic component is seen in Waalbot, a wall-climbing robot that utilizes a tri-foot design for adhesion[12]. This robot used a custom printed circuit board to run the electronics and software of the robot system, but this printed circuit board was also used as the chassis of the robot, as seen in Figure 8. This was a great way to save space, reduce the mass of the robot, and assign dual purpose to the printed circuit board. However, additional issues could have been introduced by doing this. For example, the circuit board would now have to be designed to be structurally rigid to provide reliable support as the chassis, all while also holding electrical components in place and controlling the robot controls and sensory data. Overall, this project ended up being very successful. The custom circuitry provided flexible control capabilities through software and the robot operated as desired. The only improvement sought out by the team working on this project was to attempt to reduce the sizes of the electronic components, including the motors used to drive the robot in different directions. This would aid in reducing the weight of the system, thus increasing the payload capacity and battery life and reducing the power consumption.

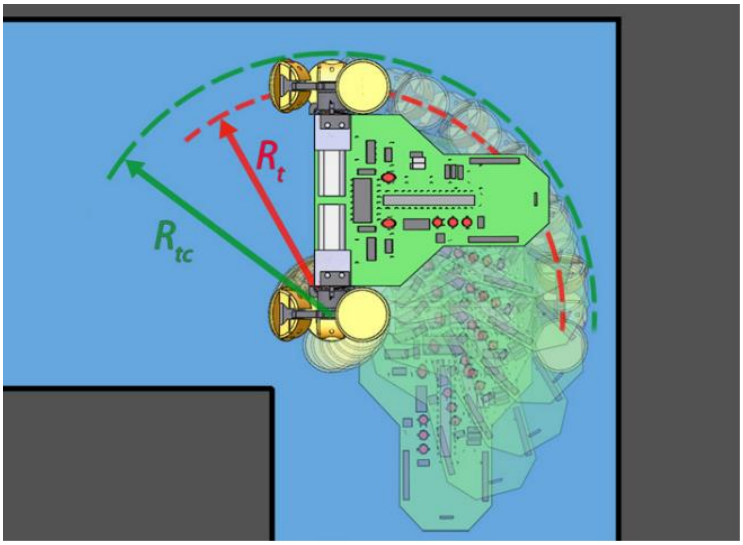

Figure 8: Waalbot tri-foot adhesion design using circuit board as robot chassis

The City-Climber robot[13] is yet another example of a successfully executed wall-climbing robot powered using suction. This particular robot, seen in Figure 9, implemented a variety of sensors in order to acquire all sorts of data from the robot. Four motors were used to control the robot's motion. An encoder was placed on each motor in order to determine the current speed and position of the robot. Pressure sensors were also used to actively measure the pressure inside the vacuum chamber. Infrared sensors measured the proximity of nearby objects to provide basic collision avoidance. Wireless control was also achieved to control the robot's onboard custom circuitry and data acquisition system. A live video stream was integrated into the system as well, providing the robot operator with a forward view of the robot's trajectory. The City-Climber was very successful in climbing walls of all types, including walls of uneven surfaces, all while maintaining a relatively small size and efficient closed-loop control system. This robot, so far, utilizes the electrical and software architecture that most successfully achieves the specifications of the robot we have developed.

Figure 9: Suction powered City-Climber robot design operating on an uneven surface

While researching currently designed wall-climbing robot prototypes, it was discovered that there are at least two companies that actively manufacture wall-climbing robots. The first is SRI International, a company that designs robots utilizing electroadhesive technology as the chosen method of adhesion[14]. These robots are controlled using wireless transmission at as far away as 100 meters and can successfully operate on nearly all surfaces, including rough and uneven surfaces. Currently, the robots are rated with a battery life that will last for about 200 meters of travel. One of these robots, pictured in Figure 10, only weighs around 1.2 kilograms and has a footprint of approximately 50mm x 70mm, making it a very small, yet successful, robot design.

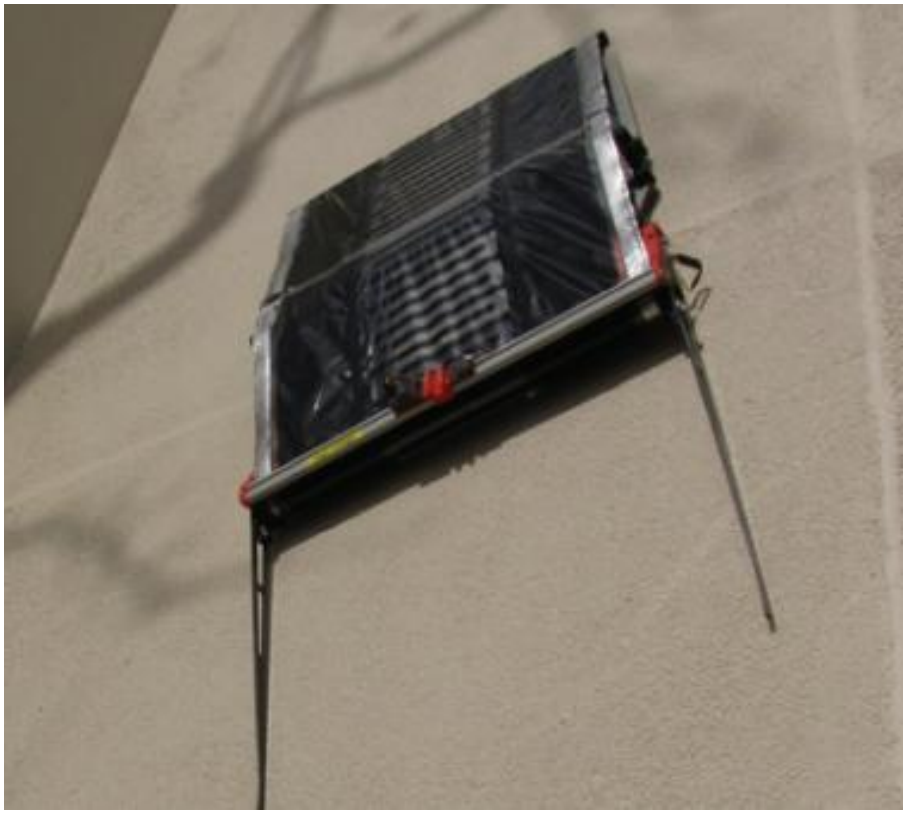

Figure 10: One of SRI International's manufactured robots available for public purchase

The other company that manufactures wall-climbing robots is called Tandemech Engineering. This company, started at California Polytechnic State University, San Luis Obispo, has developed robots that utilize custom circuitry and wireless control. The company's second prototype, Numo[15], seen in Figure 11, is a semi-autonomous robot with a very elegant exterior design. The fact that no wires or mechanical components can be seen makes the robot aesthetically appealing. The adhesion system is compared to one of a hand-held vacuum but with the addition of a flexible seal, allowing the robot to operate on nearly all surfaces. With a battery life of about 25 minutes using large rechargeable batteries, the robot is capable of a wide range of motion, including 360 degree turns. Currently, this robot is considered to be the ideal solution to this project at this point, due to its flawless functionality and appearance.

Figure 11: Tandemech Engineering's second semi-autonomous prototyped robot, Numo

III. DESIGN AND IMPLEMENTATION

Actuators

Based on the existing robot frame design, two motors were necessary to drive the front two wheels separately, allowing for turning. The motors that were first selected were DC gearmotors from Parallax[16]. These motors provided about 25 in-lb of torque at full load and had an integrated gearbox with a 30:1 input-output ratio and a quadrature encoder. They initially appeared to exceed the preliminary torque requirements determined by Stefani[6], while still having a relatively high output velocity. However, with more detailed calculation, these motors were found to not provide enough torque to drive the robot up a wall with the weight of the robot frame and preliminary electronics. Other motors were researched and Molon DC CHM model gearmotors[17] were selected. Table 1 contains information regarding the key electrical specifications of these Molon gearmotors. With an output torque of 50 in-lb, these motors exceeded the new 35 in-lb torque requirement and were capable of driving the robot at a reasonable speed. These motors, unlike the Parallax motors, did not have an integrated encoder, requiring the use of an external encoder. Additionally, the torque or performance curves for these motors were not provided, necessitating motor testing to be done to better characterize these motors.

Table 1: Key specifications for Molon DC gearmotors selected for use

Parameter	Value
Max. Operating Voltage	24 Volts
Full Load Output Speed	25 RPM
Full Load Current	1.08 Amps
Max. Motor Torque	50 in-lb

Sensors

The pressure sensors selected to monitor the various segments of the vacuum manifold were differential pressure sensors from Freescale Semiconductor, model MPX5100DP[18]. These sensors were chosen, primarily, because they were capable of measuring the pressure differential between standard atmospheric pressure and the vacuum pressure expected during operation. It was known that a perfect vacuum would not be attained due to the inherent presence of leaks in the system, so an operating range of 100 kilopascals instead of the ideal pressure differential of 101.325 kilopascals was found to be sufficient. They also featured a fully scaled and conditioned output voltage, which meant no additional circuitry was required to amplify or filter their output. Additionally, these sensors were temperature compensated, meaning that the final sensor output would not change as a function of the ambient temperature. This was a valuable feature because the robot would need to operate consistently in any outdoor and indoor environments. In the system, the positive pressure port would be open to the atmosphere, while the vacuum port was connected to the manifold air supply lines. This would output the gage pressure of the vacuum area within the manifold system. The sensor response time was about 1 millisecond, meaning the sensor would need to be sampled at time intervals larger than 1 millisecond. Table 2 contains detailed information on the important MPX5100DP sensor specifications.

Table 2: Key electrical differential vacuum pressure sensor specifications

Parameter	Value
Pressure Range	0 – 100 kPa
Supply Voltage	5.0 Volts
Supply Current	7.0 mAmps DC
Max. Voltage Output	4.7 Volts
Response Time	1.0 ms

The proximity sensors selected to integrate collision avoidance were infrared distance sensors from Sharp[19], which utilized the method of triangulation to measure object distance. These sensors were chosen because they were a simple, low-cost means of detecting objects in the robot's path during operation. Since the maximum output speed of the selected gearmotors was 25 RPM, the robot would theoretically travel a maximum of 3.5 inches per second, so the proximity sensors did not need to sense objects from far away. However, the output behavior was highly non-linear, as seen in Figure 12. Because of this, the sensors needed to be tested to verify the behavior predicted by the sensor documentation. Also, the field of view or surface reflectivity characteristics were not provided, necessitating these to be tested as well. Table 3 contains information on the key proximity sensor specifications.

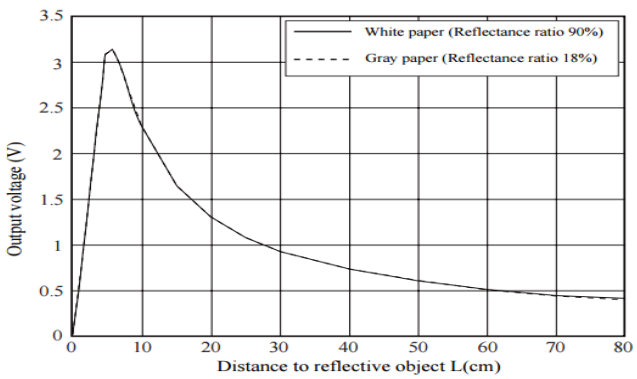

Figure 12: Non-linear output voltage versus distance of object located in front of Sharp proximity sensor

Table 3: Key specifications for Sharp distance sensor used for collision avoidance

Parameter	Value
Distance Range	10 – 80 cm
Supply Voltage	5.0 Volts
Supply Current	30 mAmps DC
Max. Voltage Output	5.3 Volts

The encoders selected to track the incremental motion of the gearmotor output shafts were Hall Effect gear tooth sensors from Hamlin[20]. Due to limited available space located on the ends of the output shafts of the gearmotors, it was difficult to directly couple quadrature encoders. It was decided that placing a small ferrous gear on empty portions of the output shafts would work best. The Hall Effect gear tooth sensors were placed within 5 millimeters of gear teeth, as shown in Figure 13, so that the sensors would detect when a gear tooth had passed by using magnetism. A square wave sensor output would then provide information about when a gear tooth passed by and how much time elapsed between each gear tooth sensed. Therefore, these sensors provided output shaft speed through software calculation using the rotational and linear resolutions of the 32-tooth gear implemented with the already designed mechanical drive system. Table 4 details the key specifications for these sensors.

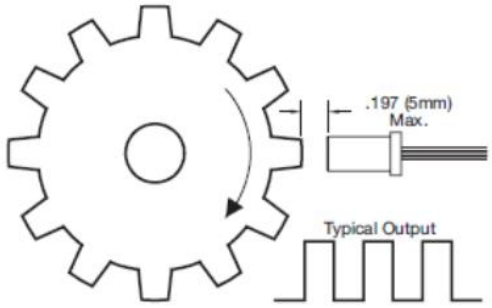

Figure 13: Setup configuration and typical output signal for a Hall Effect gear tooth sensor used as an output shaft encoder

Table 4: Key electrical specifications for Hamlin gear tooth sensor used as encoder

Parameter	Value
Max. Switching Speed	15 kHz
Supply Voltage	5.0 Volts
Supply Current	10.5 mAmps DC
Max. Voltage Output	3.0 Volts

Circuit Board

After researching the various data acquisition methods used in other robotic systems, it was decided that designing a custom circuit board would best fit the needs of this project, especially since making the electrical system versatile and adaptable for possible future additions was very important. Several specifications were used in the development of the robot's circuit board. These specifications, seen in Table 5, were determined by the desired system performance and the requirements of the sensors selected for use. The manufactured circuit board wiring diagrams and schematic can be found in Appendix A.

Table 5: Electrical specifications for wall-climbing robot system

Specification	Value	Notes
Max. Sensor Sample Rate	1 kHz	Slowest sampling rate for sensors (pressure sensors)
Min. Analog Input Pins	10	Inputs for 8 pressure sensors and 2 proximity sensors
Min. Digital Input Pins	2	Inputs for 2 Hall Effect gear tooth sensors
Min. Serial Ports	2	Ports for wired debugging & wireless communication

The ATMega1281 microcontroller[21] was selected to run the main circuit board and software operations. This microcontroller connected to all of the major subsystems of the circuit board to control, monitor, and post-process the key components and outputs. This chip included more digital pins and serial ports than necessary, which provided the option for future electronic additions. It was setup to be programmed using an in-circuit serial programming (ISP) interface. A 16 MHz clock crystal was selected to function as the primary clock source for the microcontroller. This clock crystal circuit provided the microcontroller with an external clock, which was more reliable than using the microcontroller's internal clock and improved the internal processor performance.

Two voltage regulators were required to stabilize the analog and digital power lines for the microcontroller. Since the selected component on the board and the sensors all required a 5 Volt power supply, 5 Volt constant output regulators were chosen. The chosen regulators were from Texas Instruments (model number LM340MP-05)[22]. Table 6 contains additional information on the electrical specifications of these components. The power system contained these two fixed output voltage regulators, a reverse voltage protection diode, and four decoupling capacitors.

Table 6: Analog and digital voltage regulator electrical specifications

Parameter	Value
Voltage Output	5 Volts
Current Output	1 Amps
Voltage Input	7.5 – 35 Volts

A 16 channel multiplexer was integrated into the circuit board design to provide additional analog to digital conversion input (ADC) pins. The ATMega1281 microcontroller only included eight onboard ADC pins, while ten pins were required to monitor two proximity sensors and eight pressure sensors. A 16 channel multiplexer was chosen over multiplexers with less channels to allow for the addition of other analog signals in the future. The multiplexer was used to interface four of the pressure sensors and the two proximity sensors with the microcontroller's analog to digital converter, while the other four pressure sensors were connected directly to the onboard microcontroller ADC pins. The multiplexer, therefore, accepted several analog voltage outputs and switched between these outputs, sending these analog signals through a common output line to the microcontroller.

The motor drivers selected to run the Molon gearmotors were the model VNH5019ATR-E motor drivers produced by STMicroelectronics[23]. These drivers were selected because the maximum rated voltage and current specifications exceeded the required voltage and current values needed to operate the motors. They also could handle the use of more powerful motors in the future, if necessary. Table 7 details the key electrical specifications of these drivers. The motor driver system contained two VNH5019ATR-E motor driver chips, used to control the two Molon gearmotors. It also included several resistors, used to protect the pins of the microcontroller, and reverse voltage protection circuitry. Two six-pin male headers supplied voltage to the motors and encoders and included a digital input for the encoder signals. Wide board traces were used for transporting the motor voltages and currents to prevent the traces from melting during full operation.

Table 7: STMicroelectronics motor driver chip electrical specifications

Parameter	Value
Max Supply Voltage	41 Volts
Maximum Output Current	30 Amps

The pressure sensor array used to connect the differential vacuum pressure sensors to the microcontroller contained very little circuitry compared to other subsystems of the circuit board design. There were two groups of four sensors, which were each used on opposite sides of the robot. Each sensor had a small capacitor used as a filter for the output. The sensors were all powered by the analog voltage output from the microcontroller, which was also filtered by a pair of capacitors.

It was determined that a Bluetooth communication would be ideal to wirelessly control the robot and transmit data gathered during operation. A BlueSMiRF Silver module[24] was selected for its ease of use and low cost. Table 8 contains detailed information about the Bluetooth module. The module was mounted to a breakout board with a six-pin header and contained all the necessary components to run the onboard Bluetooth modem. The header supplied the module with power and connected the module's serial communication pins with those of the microcontroller.

Table 8: Important BlueSMiRF Silver Bluetooth module electrical specifications

Parameter	Value
Max. Transmission Distance	18 meters
Operating Frequency Range	2.4 – 2.524 GHz
Serial Communication Rate Range	2400 – 115200 bps

To facilitate the debugging of the software programmed onto the microcontroller, an FT232RL USB to serial interface chip[25] and a female USB mini-B SMD connector were added to the board. This debugging interface was used to enable communication between the microcontroller and an external computer to debug the software running on the microcontroller. It converted USB data, which the computer understood, to serial USART data, which the microcontroller understood. The interface also included a pair of status LED lights and a ferrite bead with capacitors, used as filters. The circuit was configured to be powered by the computer's USB port upon successful plug-in, which allowed it to be debugged without the use of any of the other circuit board power sources and without potentially interfering with any of the board processes.

Software Code

The preliminary framework for the software code was designed, created, and provided by Professor John Ridgely. The details of each task set are discussed below, along with how each portion of the code contributed to the code's overall functionality. The C++ code can be found in Appendix B. The overall task diagram for the software can be found in Appendix C. The various equations integrated into the code for various calculations can be found in Appendix D.

The main task, *robot_main*, functioned as a setup for all of the code. The shared variables were created and the serial port was configured. One serial port was used to allow for wireless Bluetooth communication at a baud rate of 9600, meaning the serial port could send or receive up to 9600 bits of information per second. Next, all of the tasks were created, specifying their names and priorities, in particular. A priority of 1 designated the lowest priority. Finally, the scheduler was started. The scheduler ensured all tasks were performed at appropriate times based on their assigned priorities and timing specifications.

The motor task set consisted of two task files: *my_motor_task* and *motor*. These two tasks worked together to provide the two gearmotors with full functionality. The motor speeds, default rotation directions, and modes were initialized such that both gearmotors would be in brake mode upon system startup. In the robot, the gearmotors were not oriented in the same way, but instead were mirrored in order to drive the treads on both sides of the system in the same direction. Therefore, the speed value for gearmotor B was negative to accommodate the fact that gearmotor B's forward rotation occurred under the opposite settings as gearmotor A.

Then, pulse width modulation (PWM) was setup to power the gearmotors with through the microcontroller. Based on the current motor modes (1 for braking and 0 for powering) and status flag conditions, the appropriate method in *motor* was called and ran. A delay method was used to ensure that the task ran through this process once every 0.1 seconds, which allowed other tasks to run at other times. The methods for setting the motor power, reversing motor direction, increasing motor power, decreasing motor power, and motor braking were all defined and ran appropriately depending on user input. Turning was achieved using skid steering, during which one of the motors would reverse direction while the other continued to operate straight, both operating at the same speed. To turn left, for example, the left motor was reversed while the right motor remained straight, resulting in a turn radius of zero.

The encoder task set consisted of two task files: *my_encoder_task* and *encoder*. These two tasks worked together to provide the two motor encoders with full functionality. The methods for calculating current linear position and zeroing current linear position were defined. These methods were ran using conditional statements, where user input dictated the methods to be called. Two interrupt service routines (ISR) were created, one to accommodate each of the digital square wave output signals from the encoders. Each ISR ran every time an encoder pulse occurred from either encoder. The ISR's interrupted the currently running code and serviced the rising edge pulses generated by the encoders for the gearmotors. A rising edge indicated that a gear tooth had reached the Hall Effect sensor, while a falling edge indicated that the gear tooth had finished passing the sensor. Since this task's timing was set such that the code was ran through once every 0.1 seconds, the change in the amount of encoder pulses from one loop to the next was be used to calculate the current motor speeds.

The sensor task set consisted of two task files: *task_sensor* and *adc*. These two tasks worked together to provide full functionality for reading the analog outputs of eight differential vacuum pressure sensors and two proximity sensors. In *adc*, the analog to digital conversion (ADC) features of the microcontroller were enabled in order to obtain the analog output signals from these sensors. The methods for reading one analog output value and reading and averaging several analog output values were defined. In *task_sensor*, the *adc* methods were called in order to convert the analog outputs from the multiplexer channels and the microcontroller analog inputs to digital outputs. The results from the method call were used to calculate the voltage associated with the ADC reading, which represented a voltage value between 0 and 5 Volts, the analog voltage supplied by the microcontroller. The same sequence of events happened for the first four pressure sensors, which were directly connected to the ADC pins of the microcontroller. However, for the other six sensors that were connected to the multiplexer, a slightly different approach was taken. First, the appropriate channel-select pins needed to be selected to read the desired analog value from the multiplexer. All of the analog outputs from the multiplexer were sent to one of the ADC pins of the microcontroller.

There was only one user interface task used in the software design: *task_user*. This task took in user inputs and handled them to perform the desired tasks. The code used specific user serial commands to perform various tasks, such as power the motors. This task used a case structure, allowing it to accept a large amount of different user inputs and handle them all appropriately.

Graphical User Interface

A clean graphical user interface was desired for this project so that manual operation of the robot would not require interaction with the code at all. Due to the large quantity of data to be transmitted, including real-time pressure, proximity, and encoder data, it was preferred to use a Bluetooth enabled computer. MATLAB was selected to be the graphical user interface program, especially because of the integrated Bluetooth serial communication functionalities and the strong data calculation and processing capabilities of the program itself. Several features needed to be included in the interface in order for the user to have full control of the robot behavior and monitor the current state of the robot, including the speed and Bluetooth connection status, as shown in Figure 14. Motor control buttons were implemented and could be controlled using keyboard inputs, as well. Bluetooth connection buttons and a status indicator were included, also. Pressure sensor data and current robot position were plotted real-time, while current robot speed and proximity to an object ahead were updated as well. The MATLAB code for this graphical user interface can be found in Appendix E.

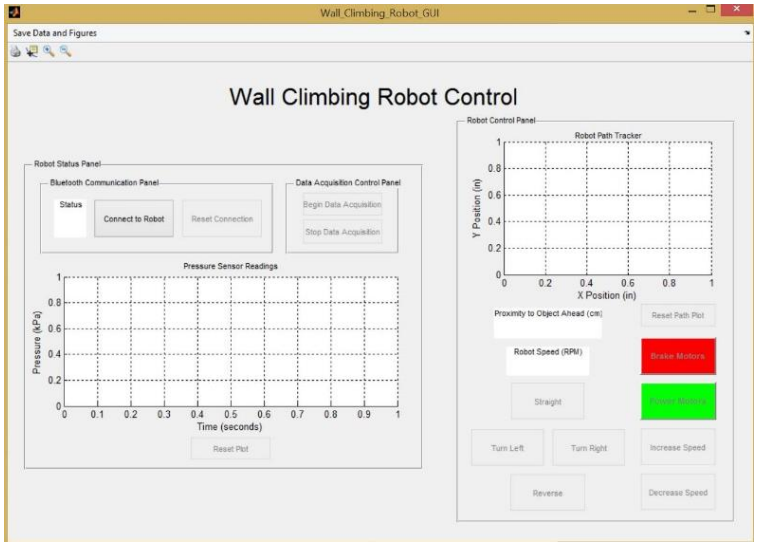

Figure 14: MATLAB graphical user control interface with buttons and data readouts

IV. TESTING AND RESULTS

Circuit Board Testing

The circuit board was surprisingly bug-free, with only a few bugs being discovered and resolved. The final board layout can be seen in Figure 15. The revised circuit board wiring diagrams and schematic can be found in Appendix A. The first issue with the board was with the analog voltage regulator. An attempt to mirror the analog voltage regulator on the board schematic resulted in the input and output pins for this regulator being swapped, meaning the supply voltage for the board was being placed on the output pin of the regulator. This caused an unregulated analog voltage to be passed to the microcontroller and analog sensors, which, thankfully, did not damage any components. To resolve the issue, the analog voltage regulator was unsoldered from the board and re-soldered in the proper orientation. The ground pin of the regulator was connected to the board ground using a jumper wire.

Due to a misreading of the microcontroller datasheet, some of the pins on the in-circuit serial programming (ISP) header were connected to incorrect pins on the microcontroller. Initially, the MISO (master-in-slave-out) and MOSI (master-out-slave-in) pins on the ISP header were connected to the MISO and MISI pins of the microcontroller, which was correct for all ATMega microcontrollers without 64 pins. This was not correct, however, for the 64 pin ATMega1281, which required the MISO and MOSI lines on the programmer to be connected to the PDO and PDI (program and debug interface) pins of the microcontroller. To resolve this issue on the board, the MISO and MOSI traces were cut and the traces still connected to the header were then spliced onto the PDO and PDI traces.

Lastly, it was discovered that the microcontroller would not program. Using an oscilloscope, the ISP pins were observed while the programmer attempted to communicate with the microcontroller. It was seen that while the reset and serial clock lines were performing as expected, the MISO and MOSI lines of the programmer incorrectly remained at logic low while the programming attempt was made. It was suggested that the FT232RL USB to serial interface chip, also connected to the MISO and MOSI lines of the programmer, were causing the issue. Therefore, the traces between the FT232RL and the microcontroller were cut. Although this resolved the programming issue, it still prevented the debugging interface from operating. As a solution, a pair of switches were spliced between the FT232RL and the microcontroller to allow for both programming and debugging to occur.

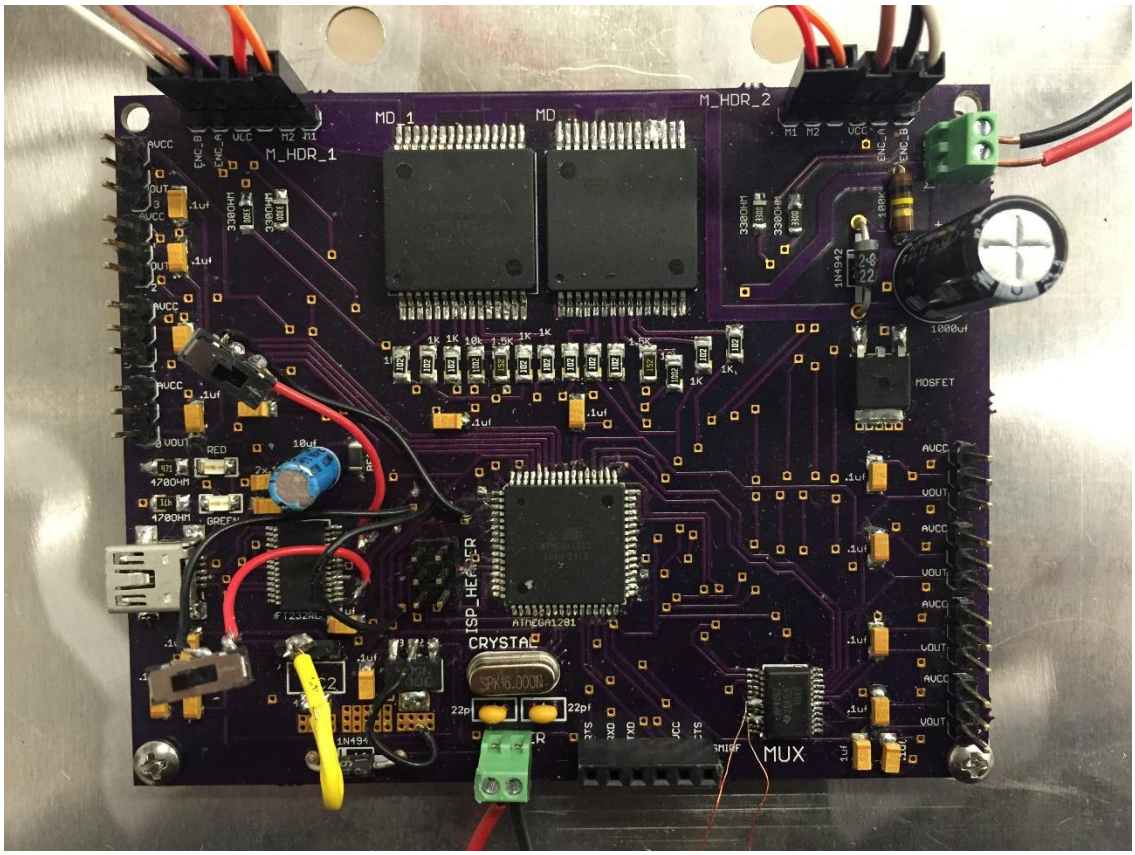

Figure 15: Final circuit board layout after circuitry issues were resolved

Pressure Sensor Testing

The differential pressure measured by the pressure sensors were calculated in kilopascals using the transfer function provided in the sensor datasheet[18]. The measured differential pressure was subtracted from atmospheric pressure and an offset value was added or subtracted from the pressure measurement so that the sensors output atmospheric pressure when there was no active suction. These calculations can be found in Appendix F.

Upon testing the system with the calibrated sensors, the steady state pressure outputs for all eight pressure sensors were stable at atmospheric pressures of around 101.3 – 101.4 kilopascals. However, when the circuit board was powered up and initialized and data acquisition was started, pressure sensor 5 did not initially start off at atmospheric pressure. Instead, the output from sensor 5 started around 106 kilopascals and slowly decreased back down to atmospheric pressure over a period of about a minute and a half, as seen in Figure 16. It appeared as if the sensor 5 pin on the multiplexer experienced some sort of voltage charge upon circuit board startup and then slowly discharged back to the voltage corresponding to the atmospheric pressure output. The reason as to why this voltage charge occurred for this sensor was because the unused pins on the sensor were touching one another, causing an undesired 5 Volt contact. The root of this issue was based around how the sensors were connected to the board and how the unused pins were not insulated or isolated, so the problem was quickly fixed. However, it was still directed that the system be given time to reach steady state prior to beginning data acquisition. Otherwise, the pressure sensors effectively measured drops in pressure to the vacuum port of the sensor and returned back to atmospheric pressure rapidly.

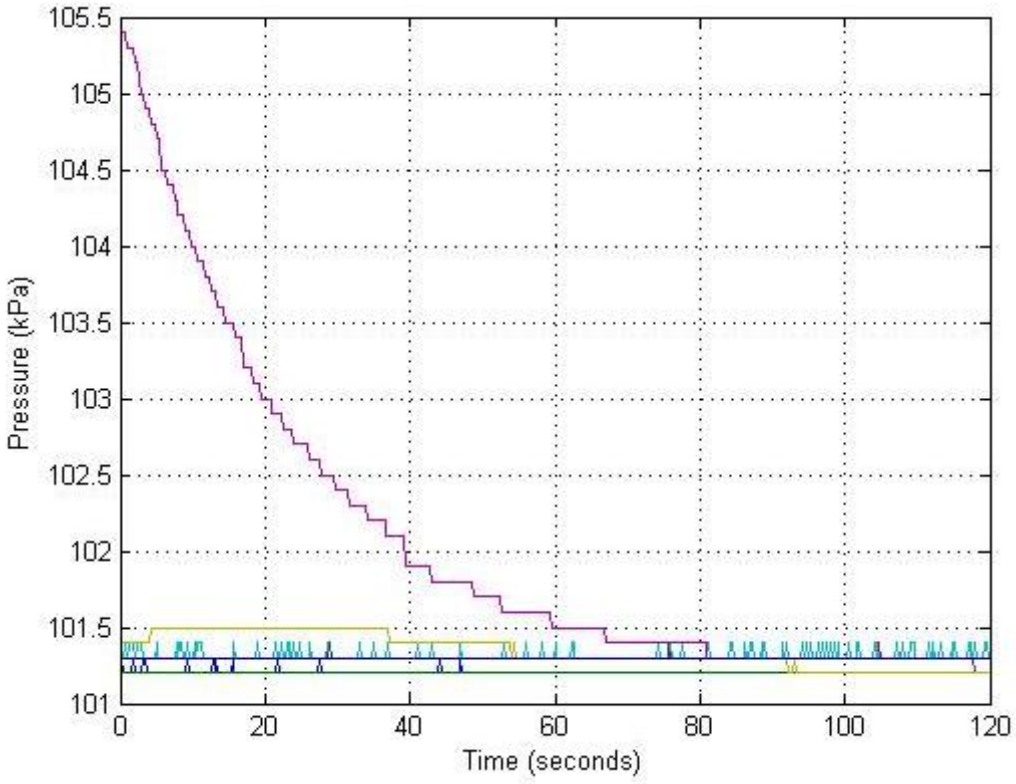

Figure 16: Pressure sensor output readings demonstrating sensor 5 voltage discharge and settling behavior upon circuit board start up

Proximity Sensor Testing

The accuracy, range, and repeatability of the proximity sensors were tested and analyzed in order to determine if basic collision avoidance could effectively be added using these sensors to provide additional measures of safety and precaution during the robot's operation. Malheiros et al.[26] detailed the testing and results of several experiments performed on the Sharp proximity sensor. The predicted results for the output of these proximity sensors were mainly based on the voltage versus distance regression models found from these experiments. The performance of these sensors was tested in a variety of different ways and a non-linear regression model for the output voltage as a function of the object distance was

derived from the experimental data. The results of this project's testing were also compared to the predicted sensor output behavior provided in the sensor datasheet[19]. The test plan, results, equations, and detailed analysis for this testing can be found in Appendix F.

The data obtained from the repeatability test was directly compared to the theoretical and datasheet models. As seen in Figure 17, the left and right sensor results followed very similar trends, yet were slightly offset from one another, with the left sensor consistently producing higher results. The theoretical model most closely matched the behavior of the left sensor, while still being between the left and right sensor outputs. The datasheet model closely followed the theoretical model trend down until around 10 centimeters, under which the datasheet model predicted a peak in the sensor output that was not tested for either sensor because only the specified measurement range of 10 – 80 centimeters was tested.

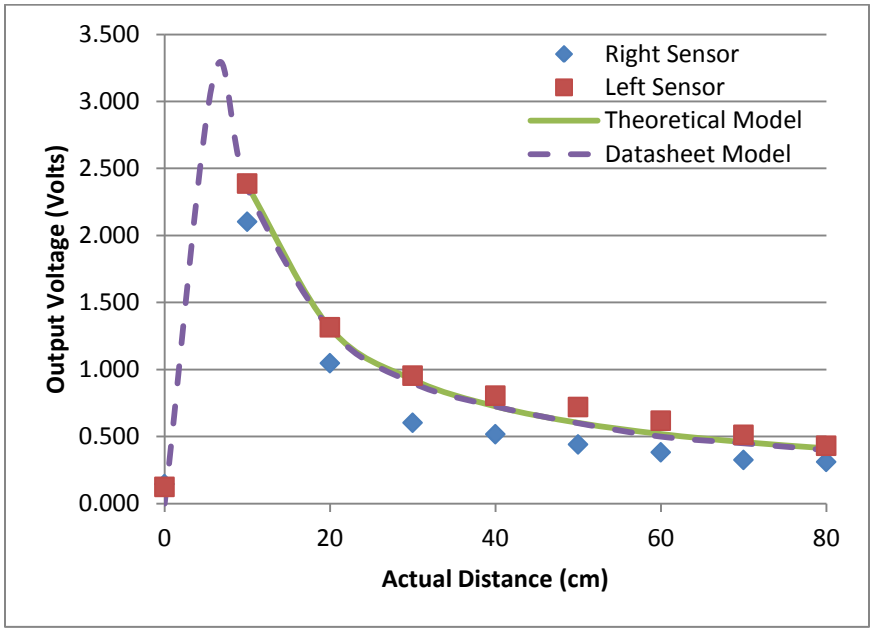

Figure 17: Left and right sensor output results from repeatability test compared directly to the theoretical model and the predicted sensor output behavior from the sensor datasheet

Overall, due to the sensors' relatively large sensitivity to object size, object orientation, and sensor orientation, as detailed in Appendix F, the sensors have been deemed less than ideal for use on the robot. The findings also significantly aided in the decision as to where on the robot the sensors would be placed in order to most effectively execute collision avoidance. Originally, it was desired to place one proximity sensor on both the front and back to the robot in order to detect possible objects in the way for forward and reverse motion. However, the analysis performed for this project demonstrated that one sensor may not be able to reliably detect any object that the robot may collide with, mainly due to the field of view restrictions on the emitter side of the sensor and the object limitations. For this reason, it was decided to place both of the proximity sensors on the front of the robot, with the receivers on the outside edges, in order to widen the overall field of view in front of the robot. Unfortunately, this took away any collision avoidance capabilities behind the robot during reverse motions. If this new configuration proved to be successful, however, it may be desired to either purchase two more of these sensors to place on the back of the robot to reinstate the collision avoidance for reverse motions or to seek other solutions to achieving reliable collision avoidance.

Bluetooth Testing

The Bluetooth module functionality for this project was very crucial to ensuring effective data transmission between the microcontroller and the graphical user interface. Essentially, in order to test the operation Bluetooth module, the Bluetooth module just needed to be used with a serial terminal prior to integrating it with the interface. The module was able to send and receive serial data and commands at a rate of 0.1 seconds with ease and no errors.

Graphical User Interface Testing

Once the graphical user interface was developed in MATLAB, all of the buttons were tested first. When a button was pressed, MATLAB was set to send a serial command to the Bluetooth module, which initiated various functionalities of the circuit board, including powering and braking the motors. Next, the Bluetooth connectivity functionalities were tested by ensuring that the "Connect to Robot" and "Reset Connection" buttons worked properly. Occasionally, MATLAB would have issues connecting to the Bluetooth module and would not start up the rest of the interface. In order to remedy this, the Bluetooth connection with the computer was terminated and refreshed, then the buttons were functional. The pressure sensor reading plot was tested as well. One issue arose from time to time when data was collected, the plot was reset, and the data acquisition was started again. When the second data collection was started, the plot would plot previously collected data overlapped with the currently collected data. However, by updating the means by which the GUI cleared and collected data, the problem was resolved. The updating of the current robot speed value worked effectively. Also, the path tracker plot worked well and plotted the robot's current and previous positions on a coordinate plane. The path tracker appeared to accurately extrapolate the motions that the motors would have created for the robot, but additional testing still needed to be done in order to prove the accuracy of this position tracking. The distance outputs of the two proximity sensors were calculated using the theoretical proximity sensor model within the software, then collected and displayed by the interface. All of the data that was collected and transmitted to the MATLAB graphical user interface was continuously stored and exported to the MATLAB workspace when data collection was completed. This allowed for additional post-processing to occur as desired.

V. CONCLUSION

Improvements

Overall, the development of this electronic system was quite successful. The robot was fully functional and could be easily controlled using the graphical user interface. Additionally, the robot position, speed, pressure values, and object proximity data were effectively and accurately updated within the interface so that the user was able to monitor the current status of the robot's various subsystems. However, several things could be done to improve the overall performance of the system. Primarily, a more robust collision avoidance system should be implemented. Although the proximity sensors provide accurate object proximity measurements, the non-linearity of the sensor output introduced error into the calculation. One improvement that could be made to the graphical user interface system would be to speed up the reading of the serial data from the Bluetooth module. The "query" function was used to send the module a command to send the data to the interface and then read the data as text. However, this function requires around 0.2 seconds to run, thus limiting the data acquisition sampling period to 0.2 seconds. The internal MATLAB timer was used to obtain data every time the period time elapsed, but the timer was able to run a period as fast as 0.01 seconds. This faster period would be preferred in order to sample the pressure sensor data more quickly and enable the user to see the true behavior and status of the vacuum pressure system. The circuit board could also be reprinted and soldered to incorporate the board revisions. Lastly, a rate gyro sensor could be implemented to directly measure the angular position and velocities associated with the skid turning, replacing the indirect calculation of these values using the encoder reading and not requiring testing on the effects of tread slip.

Future Work

The electronics, software, and graphical user control interface could absolutely be further developed in the future to integrate additional functionalities into the system. In order to do so, additions may need to be made to the circuit board, most likely requiring the reprinting of the circuit board. The code would also need to be altered to accommodate any changes to the robot system, which could be easily accomplished by referencing the programming operation manual included in Appendix H.

Testing should still be done on the gearmotors once the system can be tested using the vacuum adhesion system. The motor power and torque behaviors need to be characterized and correlated to the motor power setting values set within the software. The motor testing would be crucial to verifying that the selected gearmotors are actually capable of providing the power necessary to drive the robot along a wall using the vacuum adhesion system.

Testing should also be done on the encoders to verify the accuracy of the encoder readings and calculations with the effects of slip in the treads. Theoretically, for an accurate encoder pulse, the calculated robot position and speed would be accurate. However, the calculation for the position did not factor in the effects of tread slip during operation, especially during a skid steer turn. Also, the speed calculation was dependent on the timing of each run through my_*encoder_task*, since the change in position calculated from one run to the next was divided by the designated run time of 0.1 seconds. This potential for position and speed errors necessitates the testing of the encoders during operation. The test plan and calculations to be used can be seen in Appendix F.

Solenoids could possibly also be added to the vacuum system in order to control the pressure distributions to each of the manifold sections. Continuous solenoids would be ideal, since an open/close solenoid would potentially cause the robot to entirely lose suction and fall off of the wall with inadequate timing control. However, to integrate solenoids to control the vacuum pressure within the manifold sections, a control system would need to be developed.

Work was previously done to upgrade the robot to being more autonomous. Mainly, a PID control system to be developed within the software was investigated to determine how to implement accurate autonomous positioning and speed control. These capabilities would allow the robot to accept a command for a destination position and desired speed and autonomously travel to that position at that speed using feedback from the encoders and motors. A basic motor model was created within Simulink, found in Appendix J, to observe the motor response to various combinations of PID control parameters. A controller model within could easily be implemented into the existing MATLAB interface code to obtain information regarding motor status and send commands to the microcontroller accordingly.

Lastly, ADAMS/Machinery software was also used, shown in Appendix I, in hopes of determining the behavior of the basic frame and shaft assemblies of the robot mechanical hardware as various motor controller models operate at various speeds and powers. The motor curves for any motor can be quickly input into the ADAMS/Machinery motor model in order to later analyze the motor behavior on the robot drive shafts and frame. The portion of this work that was already accomplished worked to verify that the motor and bearing models were implemented correctly in the ADAMS/Machinery model of the robot body.

BIBLIOGRAPHY

[1] Teoh, Z. E., S. B. Fuller, P. Chirarattananon, N. O. Pérez-Arancibia, J.D. Greenburg and R. J. Wood. "A Hovering Flapping-Wing Microrobot with Altitude Control and Passive Upright Stability." Harvard University, n.d. <http://www.micro.seas.harvard.edu/papers/IROS12_Teoh.pdf>

[2] Keennon, Matthew, Karl Klingebiel, Henry Won and Alexander Andriukov. "Development of the Nano Hummingbird: A Tailless Flapping Wing Micro Air Vehicle." *American Institute of Aeronautics and Astronautics* (Jan. 2012).

[3] Dethe, Raju D. and Dr. S.B. Jaju. "Developments in Wall Climbing Robots: A Review." *International Journal of Engineering Research and General Science* (April-May 2014): 33-42.

[4] Berns, K., C. Hillenbrand and T. Luksch. "Climbing Robots for Commercial Applications – a Survey." University of Kaiserslautern, n.d. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.216.287&rep=rep1&type=pdf>

[5] Kolhalkar, N. R. and S. M. Patil. "Wall Climbing Robots: A Review." *International Journal of Engineering and Innovative Technology* (May 2012): 227-229.

[6] Von Broekhoven, Erik, Jim Stefani and Thomas J. Mackin. "Sliding Tread Adhesion: A New Approach for Climbing Robots." California Polytechnic State University, San Luis Obispo, Sept. 2014.

[7] Mahajan, Ritesh G. and S. M. Patil. "Development of Wall Climbing Robot for Cleaning Application." *International Journal of Emerging Technology and Advanced Engineering* (May 2013): 658-662.

[8] Albagul, A., A. Asseni and O. Khalifa. "Wall Climbing Robot: Mechanical Design and Implementation." *Recent Advances in Circuits, Systems, Signal and Telecommunications* (n.d.): 28-32.

[9] Gostanian, Justin and Erick Read. "Design and Construction of a Tree Climbing Robot." Worcester Polytechnic Institute, Apr. 2012. <https://www.wpi.edu/Pubs/E-project/Available/E-project-042512-231554/unrestricted/TCR_MQP.pdf>

[10] Vishanth, B., S. Kathiravan, S. Giri Prasad, R. Raju and D. James Deepak. "Analysis of a Wall Climbing Robot." *International Journal of Innovative Research in Science, Engineering and Technology* (Mar. 2014): 1293-1297.

[11] Kim, Hwang, Dongmok Kim, Hojoon Yang, Kyouhee Lee, Kuchan Seo, Doyoung Chang and Jongwon Kim. "Development of a Wall-Climbing Robot Using a Tracked Wheel Mechanism." *Journal of Mechanical Science and Technology* (Apr. 2008).

[12] Unver, Ozgur, Michael P. Murphy and Metin Sitti. "Geckobot and Waalbot: Small-Scale Wall Climbing Robots." Carnegie Mellon University, n.d. <http://nanolab.me.cmu.edu/publications/papers/Unver-AIAA2005.pdf>

[13] Xiao, Jizhong and Ali Sadegh. "City-Climber: A New Generation Wall-Climbing Robots." The City College, City University of New York, n.d. <http://cdn.intechopen.com/pdfs-wm/485.pdf>

[14] SRI International. "Wall Climbing Robots – Electroadhesive Robots for Robust Vertical Mobility." <https://www.sri.com/sites/default/files/brochures/sri_wallclimbingrobots.pdf>

[15] Tandemech Engineering. "Numo." <http://www.tandemech.com/numo.html>

[16] "6-15V Gear Motor with Encoder (#28819)." Rev. 1.1. Parallax Incorporated. Jan. 2015.

[17] Molon Motor & Coil Corporation. "CHM – DC Permanent Magnet Gearmotor." <http://www.molon.com/standard_dc_motors.html>

[18] "Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated." Rev. 13. Freescale Semiconductor. May 2010.

[19] "Distance Measuring Sensor Unit." Sharp Corporation. Dec. 2006.

[20] "55505 Hall Effect Flange Mount Geartooth Sensor." Rev. AD. Hamlin. Sept. 2012.

[21] "8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash." Atmel Corporation. Oct. 2012.

[22] "LM340-N/LM78XX Series 3-Terminal Positive Regulators." Texas Instruments. Dec. 2013.

[23] "VNH5019A-E Automotive Full Integrated H-Bridge Motor Driver." Rev. 9. STMicroelectronics. Sept. 2013.

[24] "RN-42/RN-42-N Data Sheet." Rev. 1.0. Roving Networks. June 2011.

[25] "FT232 USB UART IC." Rev. 2.11. Future Technology Devices International. Apr. 2015.

[26] Malheiros, Paulo, José Goncalves and Paulo Costa. "Towards a More Accurate Infrared Distance Sensor Model." University of Porto, Portugal, n.d. <https://bibliotecadigital.ipb.pt/bitstream/10198/4119/1/iscies09_sharp_model.pdf>

APPENDICES

A. CIRCUIT BOARD WIRING DIAGRAMS AND SCHEMATICS

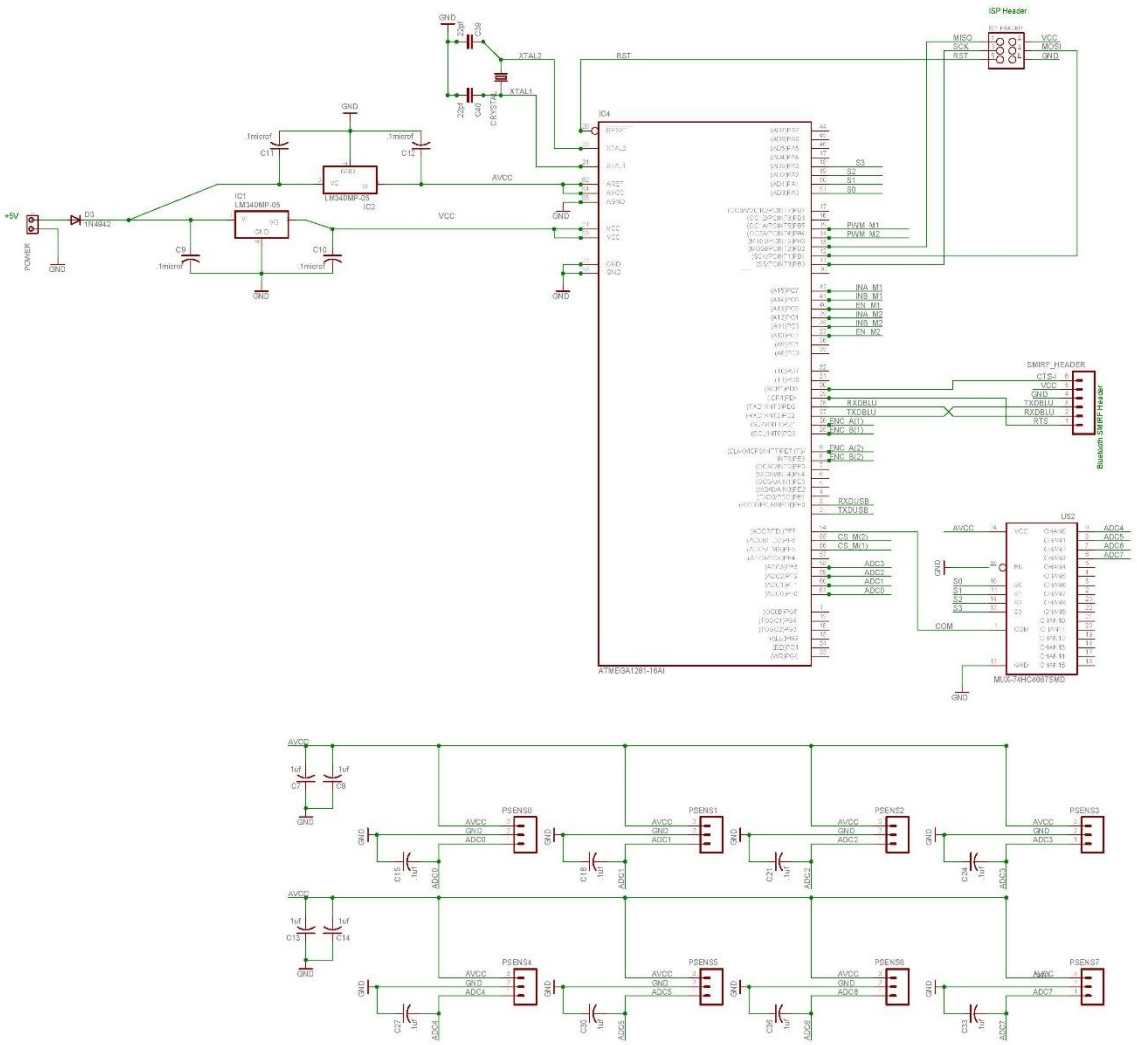

Figure 18: Manufactured circuit board wiring schematic for microcontroller, pressure sensor array, board power system, multiplexer, Bluetooth module, and in-circuit serial programming

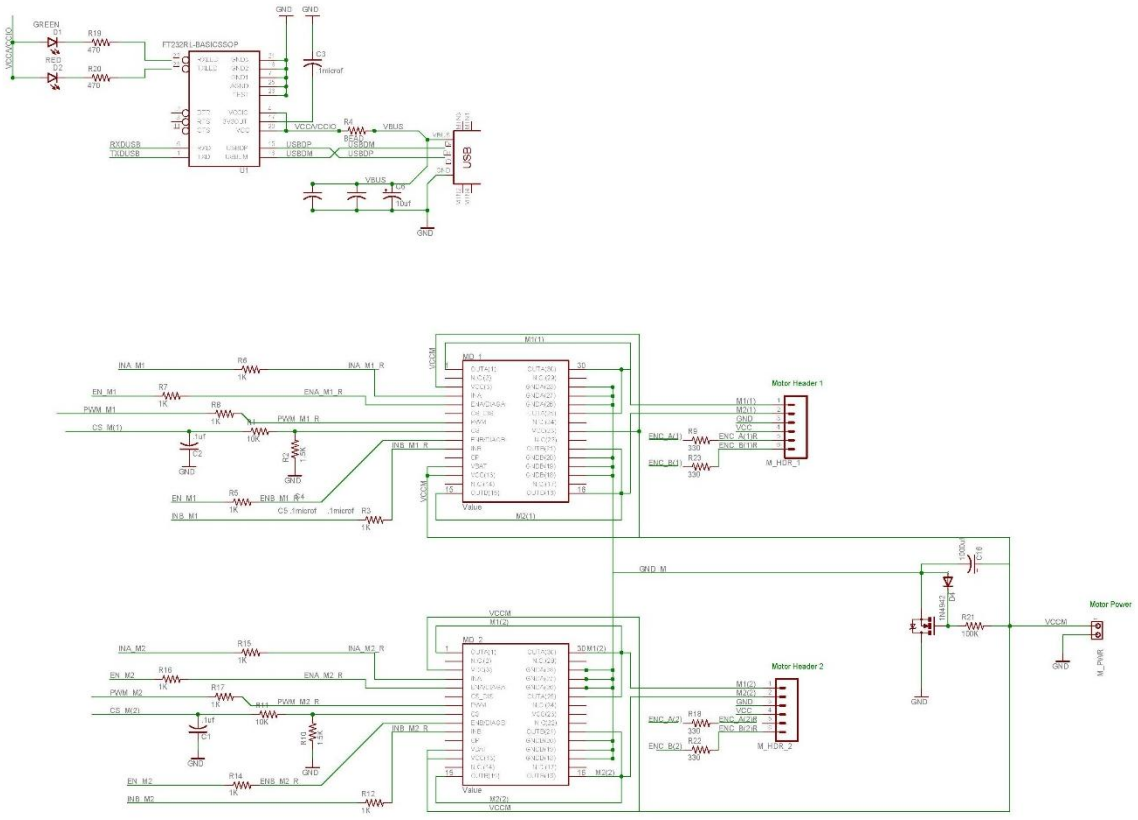

Figure 19: Manufactured circuit board wiring schematic for motor drivers and FT232RL USB to serial interface chip

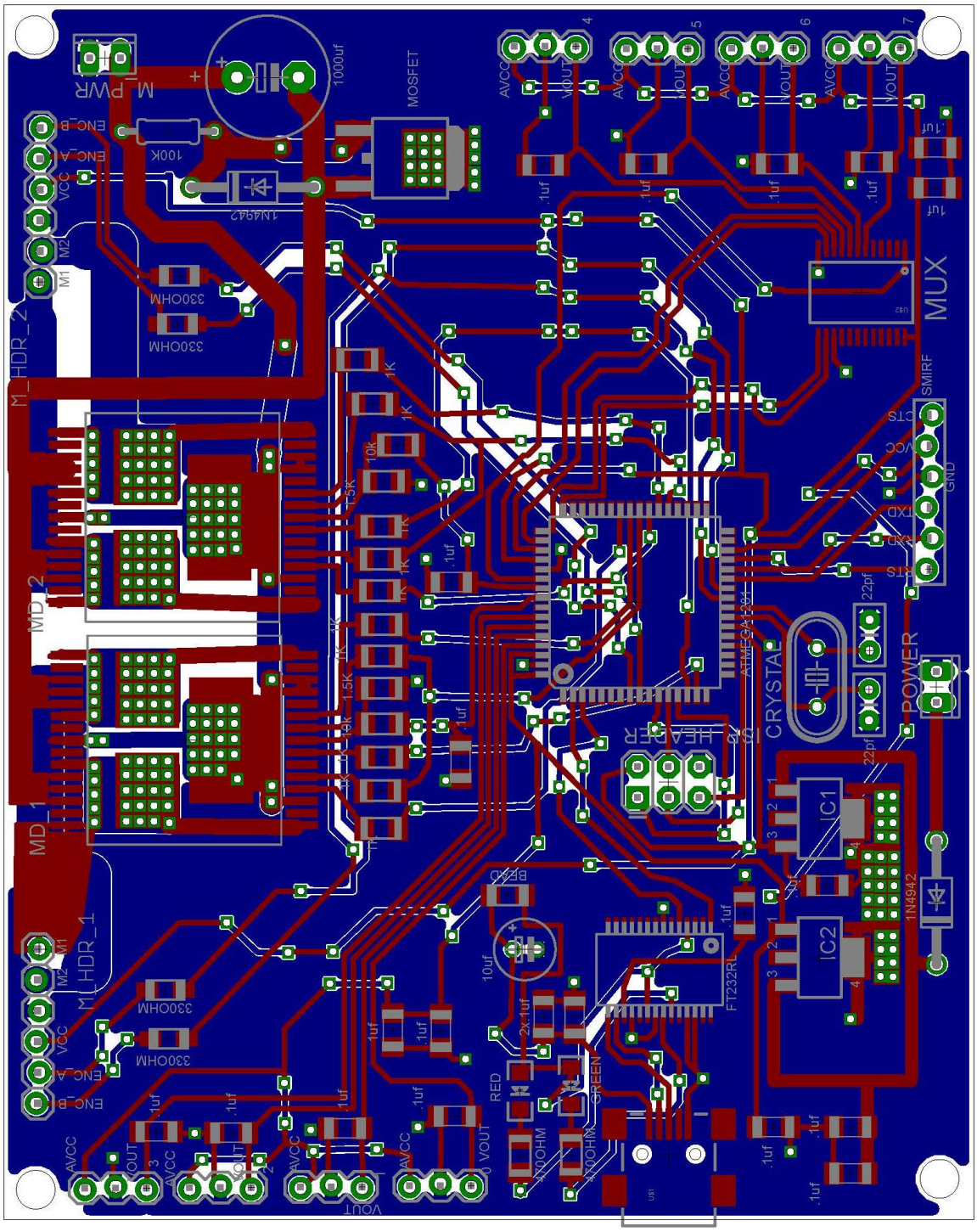

Figure 20: Manufactured circuit board schematic showing overall board layout and complete trace configuration

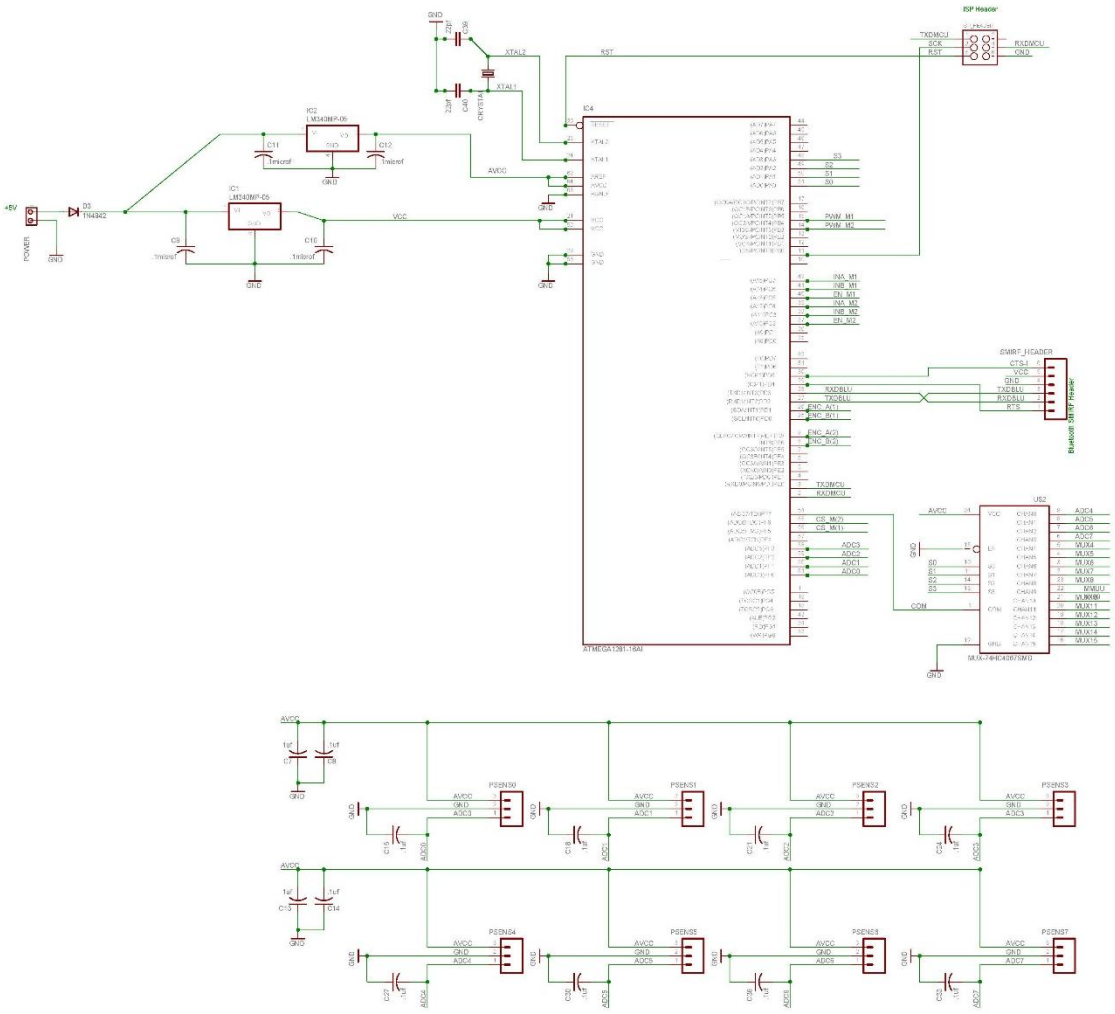

Figure 21: Revised circuit board wiring schematic for microcontroller, pressure sensor array, board power system, multiplexer, Bluetooth module, and in-circuit serial programming circuitry with corrections made based on circuit board testing findings

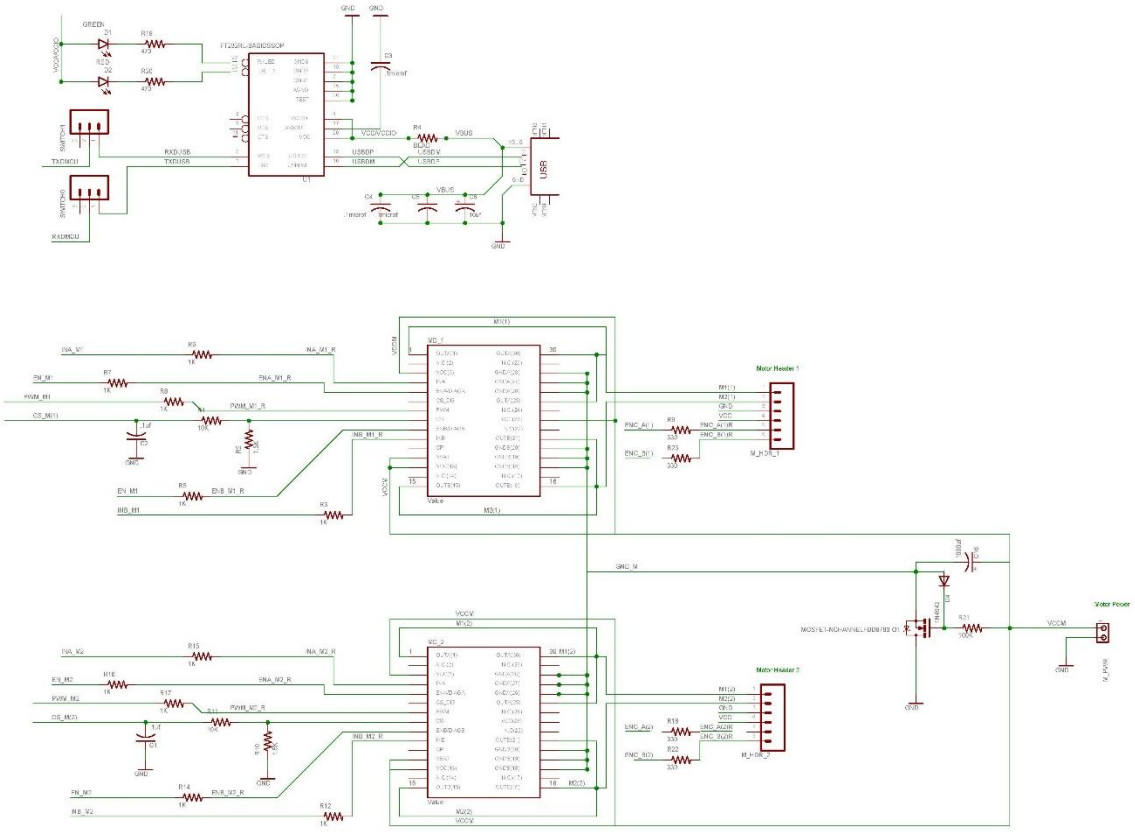

Figure 22: Revised circuit board wiring schematic showing circuitry with corrections based on circuit board testing findings for motor drivers, and FT232RL USB to serial interface chip

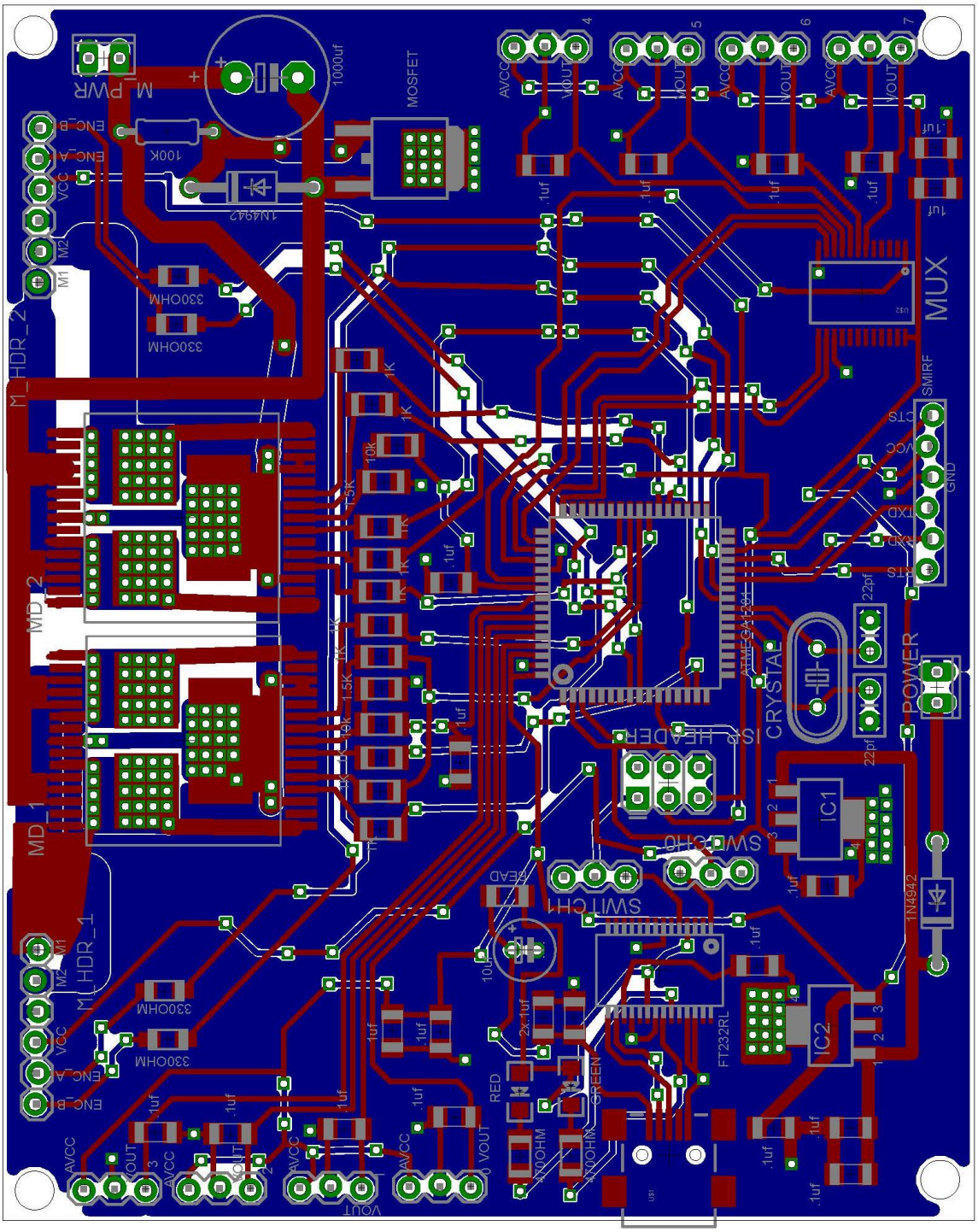

Figure 23: Revised circuit board schematic showing overall board layout and trace configuration circuitry with corrections made based on circuit board testing findings

Table 9: Detailed pin information for ATMega1281 microcontroller on circuit board

Signal	Pin #	Pin Name	Function
RST	20	RESET	Resets Microcontroller
XTAL2	23	XTAL2	Clock Input
XTAL1	24	XTAL1	Clock Output
AVCC	62/64	AREF/ACC	Analog Reference and Supply Voltage
GND	63	AGND	Analog Ground
VCC	21/52	VCC	Digital Reference and Supply Voltage
GND	22/53	GND	Digital Ground
S3	48	PA3	MUX Channel Select Pin 3
S2	49	PA2	MUX Channel Select Pin 2
S1	50	PA1	MUX Channel Select Pin 1
S0	51	PA0	MUX Channel Select Pin 0
PWM_M1	15	PB5	Motor A PWM Output
PWM_M2	14	PB4	Motor B PWM Output
SCK	11	PB1	Programming Clock Line
INA_M1	42	PC7	Motor A Power Line 1
INB_M1	41	PC6	Motor A Power Line 2
EN_M1	40	PC5	Motor A Enable
INA_M2	39	PC4	Motor B Power Line 1
INB_M2	38	PC3	Motor B Power Line 2
EN_M2	37	PC2	Motor B Enable
CTS-1	30	PD5	Bluetooth Clear to Send
RTS	29	PD4	Bluetooth Request to Send
RXDBLU	28	PD3	Data Input to Bluetooth
TXDBLU	27	PD2	Data Output from Bluetooth
ENC_A1	26	PD1	Motor A Encoder Interrupt Input
ENC_A2	9	PE7	Motor B Encoder Interrupt Input
TXDMCU	3	PE1	Data Input to Programmer
RXDMCU	2	PE0	Data Output to Programmer
COM	54	PF7	MUX Common Output
CS_M2	55	PF6	Motor B Current Sense
CS_M1	56	PF5	Motor A Current Sense
ADC3	58	PF3	Pressure Sensor 4 Reading Input
ADC2	59	PF2	Pressure Sensor 3 Reading Input
ADC1	60	PF1	Pressure Sensor 2 Reading Input
ADC0	61	PF0	Pressure Sensor 1 Reading Input

B. C++ CODE

```
//*****************************************************************************
/** \file robot_main.cpp
 *    This file contains the main code for a program which runs the custom board for
 *    the Wall Climbing Robot Thesis Project. Currently this code runs two DC motors,
 *    two hall effect encoders, a BlueSmirf Bluetooth module, 8 pressure sensors, and 2
 *    proximity sensors. It sends the sensor and motor information via Bluetooth to a
 *    MATLAB GUI that displays the data to the user. Through the MATLAB GUI, the user
 *    can also control the speed of the motors, thus controlling the overall speed and
 *    trajectory of the robot manually.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task plus a main one
 *    \li 10-30-2012 JRR A hopefully somewhat stable version with global queue
 *                       pointers and the new operator used for most memory allocation
 *    \li 11-04-2012 JRR FreeRTOS Swoop demo program changed to a sweet test suite
 *    \li 01-05-2012 JRR Program reconfigured as ME405 Lab 1 starting point
 *    \li 02-04-2014 AJB Program runs two motors
 *    \li 02-11-2014 LBT Added encoder shared data variables
 *    \li 11-29-2014 LBT Added code for bluetooth communication and task
 *    \li 12-02-2014 LBT Added shared variables for pressure sensors, reversing motor
 *                       direction flag, and increasing/decreasing motor power flag
 *    \li 12-05-2014 LBT Updated comments
 *    \li 05-12-2015 LBT Changed all serial ports to Bluetooth port and created
 *                       Bluetooth text queue, updated comments and formatting
 *    \li 05-13-2015 LBT Removed ISR shared variables for old interrupt values
 *    \li 05-20-2015 LBT Added shared variables for motor control
 *    \li 05-29-2015 LBT Added shared variables for proximity sensors
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#include <stdlib.h>                                     // Prototype declarations for I/O
functions
#include <avr/io.h>                                     // Port I/O for SFR's
#include <avr/wdt.h>                                    // Watchdog timer header
#include <string.h>                                     // Functions for C string handling
#include "FreeRTOS.h"                                   // Primary header for FreeRTOS
#include "task.h"                                       // Header for FreeRTOS task functions
#include "queue.h"                                      // FreeRTOS inter-task communication
```

```
queues
#include "croutine.h"                                   // Header for co-routines and such
#include "rs232int.h"                                   // ME405/507 library for serial comm.
#include "time_stamp.h"                                 // Class to implement a microsecond timer
#include "frt_task.h"                                   // Header of wrapper for FreeRTOS tasks
#include "frt_text_queue.h"                             // Wrapper for FreeRTOS character queues
#include "frt_queue.h"                                  // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                            // Header for thread-safe shared data
#include "task_user.h"                                  // Header for user interface task
#include "my_motor_task.h"                              // Include Motor Task functions
#include "my_encoder_task.h"                            // Include Encoder task functions
#include "shares.h"                                     // Global ('extern') queue declarations
#include "task_sensor.h"                                // Include Sensor task functions

// Declare the queues which are used by tasks to communicate with each other here.
// Each queue must also be declared 'extern' in a header file which will be read
// by every task that needs to use that queue. The format for all queues except
// the serial text printing queue is 'frt_queue<type> name (size)', where 'type'
// is the type of data in the queue and 'size' is the number of items (not neces-
// sarily bytes) which the queue can hold.

/** This is a print queue, descended from \c emstream so that things can be printed
 *  into the queue using the "<<" operator and they'll come out the other end as a
 *  stream of characters. It's used by tasks that send things to the user interface
 *  task to be printed.
 */
frt_text_queue* print_ser_queue;
frt_text_queue* bluetooth_ser_queue;

// Shared_data variables shown below
shared_data<uint8_t>*  p_modeA;                         // Motor A mode (brake, power)
shared_data<uint8_t>*  p_modeB;                         // Motor B mode (brake, power)
shared_data<uint8_t>*  p_motor;                         // Motor selection (1/A or 2/B)
shared_data<uint8_t>*  p_movement;                      // Current robot movement (0: Straight,
1: Reverse, 2: Left, 3: Right)
shared_data<uint8_t>*  p_zeropositionflag;              // Zero position and error count flag
shared_data<int32_t>*  p_countA;                        // Encoder A interrupt count
shared_data<int32_t>*  p_countB;                        // Encoder B interrupt count
shared_data<double>*   p_positionA;                     // Motor A position
shared_data<double>*   p_positionB;                     // Motor B position
shared_data<double>*   p_position;                      // Motor B position
shared_data<double>*   p_proximity;                     // Motor B position
shared_data<double>*   p_x_position;                    // Current x position
shared_data<double>*   p_y_position;                    // Current y position
shared_data<double>*   p_theta;                     // Angular position during turn
shared_data<double>*   p_theta_max;                     // Angular position during turn
shared_data<int32_t>*  p_motor_powerA;                  // Current motor A power
shared_data<int32_t>*  p_motor_powerB;                  // Current motor B power
shared_data<double>*   p_motor_speedA;                  // Current motor A speed
shared_data<double>*   p_motor_speedB;                  // Current motor B speed
shared_data<double>*   p_speed;                         // Current robot speed
shared_data<uint8_t>*  p_reverse_direction_flag;        // Reverse motor direction flag
shared_data<uint8_t>*  p_increase_power_flag;           // Increase the motor power flag
```

```
shared_data<uint8_t>*  p_decrease_power_flag;          // Decrease the motor power flag
shared_data<uint8_t>*  p_turn_left_flag;               // Turn left flag
shared_data<uint8_t>*  p_turn_right_flag;              // Turn right flag
shared_data<uint8_t>*  p_straight_flag;                // Go straight flag
shared_data<double>*   p_sensor1;                      // Pressure sensor 1 reading
shared_data<double>*   p_sensor2;                      // Pressure sensor 2 reading
shared_data<double>*   p_sensor3;                      // Pressure sensor 3 reading
shared_data<double>*   p_sensor4;                      // Pressure sensor 4 reading
shared_data<double>*   p_sensor5;                      // Pressure sensor 5 reading
shared_data<double>*   p_sensor6;                      // Pressure sensor 6 reading
shared_data<double>*   p_sensor7;                      // Pressure sensor 7 reading
shared_data<double>*   p_sensor8;                      // Pressure sensor 8 reading
shared_data<double>*   p_sensor9;                      // Proximity sensor 1 voltage reading
shared_data<double>*   p_sensor10;                     // Proximity sensor 2 voltage reading

//*****************************************************************************
/** \brief   The main function sets up the RTOS.
 *  \details Some test tasks are created. Then the scheduler is started up; the scheduler
 *           runs until power is turned off or there's a reset.
 *  @return  This is a real-time microcontroller program which doesn't return. Ever.
 */

int main (void)
{
    // Disable the watchdog timer . This is important because sometimes the watchdog
    // timer may have been left on...and it tends to stay on.
    MCUSR = 0;                                         // Resets the MCU status register
    wdt_disable ();                                    // Disable watchdog timer

    // Create shared_data variables
    p_modeA = new shared_data<uint8_t>;                // Motor A mode (freewheel, brake, power)
    p_modeB = new shared_data<uint8_t>;                // Motor B mode (freewheel, brake, power)
    p_motor = new shared_data<uint8_t>;                // Motor selection (1/A or 2/B)
    p_movement = new shared_data<uint8_t>;             // Current motor movement
    p_countA = new shared_data<int32_t>;               // Encoder A interrupt count
    p_countB = new shared_data<int32_t>;               // Encoder B interrupt count
    p_positionA = new shared_data<double>;             // Motor A position
    p_positionB = new shared_data<double>;             // Motor B position
    p_position = new shared_data<double>;              // Motor B position
    p_proximity = new shared_data<double>;             // Motor B position
    p_x_position = new shared_data<double>;            // Current x position
    p_y_position = new shared_data<double>;            // Current y position
    p_theta = new shared_data<double>;                 // Angular position during turn
    p_theta_max = new shared_data<double>;             // Angular position during turn
    p_zeropositionflag = new shared_data<uint8_t>;     // Zero error and position flag
    p_motor_powerA = new shared_data<int32_t>;         // Desired motor A power
    p_motor_powerB = new shared_data<int32_t>;         // Desired motor B power
    p_motor_speedA = new shared_data<double>;          // Desired motor A speed
    p_motor_speedB = new shared_data<double>;          // Desired motor B speed
    p_speed = new shared_data<double>;                 // Current robot speed
    p_reverse_direction_flag = new shared_data<uint8_t>;// Reverse motor direction flag
    p_increase_power_flag = new shared_data<uint8_t>;  // Increase motor power flag
    p_decrease_power_flag = new shared_data<uint8_t>;  // Decrease motor power flag
```

```
    p_turn_left_flag = new shared_data<uint8_t>;        // Turn left flag
    p_turn_right_flag = new shared_data<uint8_t>;       // Turn right flag
    p_straight_flag = new shared_data<uint8_t>;         // Go straight flag
    p_sensor1 = new shared_data<double>;                // Pressure sensor 1 reading
    p_sensor2 = new shared_data<double>;                // Pressure sensor 2 reading
    p_sensor3 = new shared_data<double>;                // Pressure sensor 3 reading
    p_sensor4 = new shared_data<double>;                // Pressure sensor 4 reading
    p_sensor5 = new shared_data<double>;                // Pressure sensor 5 reading
    p_sensor6 = new shared_data<double>;                // Pressure sensor 6 reading
    p_sensor7 = new shared_data<double>;                // Pressure sensor 7 reading
    p_sensor8 = new shared_data<double>;                // Pressure sensor 8 reading
    p_sensor9 = new shared_data<double>;                // Proximity sensor 1 voltage reading
    p_sensor10 = new shared_data<double>;               // Proximity sensor 2 voltage reading

    // Configure serial ports which can be used by a task to print debugging infor-
    // mation, or to allow user interaction, or for whatever use is appropriate.  The
    // serial ports will be used by the user interface task and sensor task after setup is
    complete and
    // the task scheduler has been started by the function vTaskStartScheduler().
    //rs232* ser_port = new rs232 (9600, 0);
    //*ser_port << clrscr << PMS ("Wall Climbing Robot - Starting Program") << endl;

    rs232* bluetooth_ser_port = new rs232 (9600, 1);
    //*bluetooth_ser_port << clrscr << PMS ("Wall Climbing Robot - Bluetooth port") << endl;

    // Create the queues and other shared data items here.
    //print_ser_queue = new frt_text_queue (32, ser_port, 10);
    bluetooth_ser_queue = new frt_text_queue (32, bluetooth_ser_port, 10);

    // The user interface is at low priority. Controls inputs and outputs to GUI.
    new task_user ("UserInterface", task_priority (1), 260, bluetooth_ser_port);

    // Create a task which reads the A/D and sensor data.
    new task_sensor ("Sensor", task_priority (2), 280, bluetooth_ser_port);

    // Creates a task that controls the motors.
    new my_motor_task ("Motors", task_priority (2), 280, bluetooth_ser_port);

    // Creates a task that reads and deciphers the encoder.
    new my_encoder_task ("Encoders", task_priority (2), 280, bluetooth_ser_port);

    // Creates a task for reading the sensor data.
    new adc ("ADC", task_priority (2), 280, bluetooth_ser_port);

    // Here's where the RTOS scheduler is started up. It should never exit as long as
    // power is on and the microcontroller isn't rebooted.
    vTaskStartScheduler ();
}
```

```
//================================================================================
/** \file shares.h
 *    This file contains extern declarations for queues and other inter-task data
 *    communication objects used in the Wall Climbing Robot project.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task plus a main one
 *    \li 10-29-2012 JRR Reorganized with global queue and shared data references
 *    \li 02-11-2014 AIB Added shared variables for encoder user input functionality
 *    \li 11-29-2014 LBT Added shared variables for second encoder and deleted some variables
 *    \li 12-02-2014 LBT Added shared variables for pressure sensors, reversing motor direction
 flag,
 *                       and increasing/decreasing motor power flags
 *    \li 05-12-2015 LBT Updated comments and formatting
 *    \li 05-13-2015 LBT Removed ISR shared variables for old interrupt values
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//================================================================================
#ifndef _SHARES_H_                  // This define prevents this .h file from being
#define _SHARES_H_                  // included multiple times in a .cpp file

//================================================================================
// Externs:  In this section, we declare variables and functions that are used in all
// (or at least two) of the files in the data acquisition project. Each of these items
// will also be declared exactly once, without the keyword 'extern', in one .cpp file
// as well as being declared extern here.

// These queues allows tasks to send characters to the user interface task for display.
extern frt_text_queue* print_ser_queue;
extern frt_text_queue* bluetooth_ser_queue;

// These variables were added to share data between the tasks.
// They can be used by any file that can see "shares.h".

extern shared_data<uint8_t>*  p_modeA;                // Motor A mode (brake, power)
extern shared_data<uint8_t>*  p_modeB;                // Motor B mode (brake, power)
extern shared_data<uint8_t>*  p_motor;                // Motor selected (1 or 2 or both)
extern shared_data<uint8_t>*  p_movement;             // Current robot movement
extern shared_data<uint8_t>*  p_zeropositionflag;     // Zero position and error count flag
```

```
extern shared_data<int32_t>*  p_countA;                 // Encoder for motor A interrupt count
extern shared_data<int32_t>*  p_countB;                 // Encoder for motor B interrupt count
extern shared_data<double>*   p_positionA;              // Encoder for motor A interrupt count
extern shared_data<double>*   p_positionB;              // Encoder for motor B interrupt count
extern shared_data<double>*   p_position;               // Encoder for motor B interrupt count
extern shared_data<double>*   p_proximity;              // Encoder for motor B interrupt count
extern shared_data<double>*   p_x_position;             // Current x position
extern shared_data<double>*   p_y_position;             // Current y position
extern shared_data<double>*   p_theta_max;              // Angular position during turn
extern shared_data<double>*   p_theta;                  // Angular position during turn
extern shared_data<int32_t>*  p_motor_powerA;           // Current motor A power
extern shared_data<int32_t>*  p_motor_powerB;           // Current motor B power
extern shared_data<double>*   p_motor_speedA;           // Current motor A speed
extern shared_data<double>*   p_motor_speedB;           // Current motor B speed
extern shared_data<double>*   p_speed;                  // Current robot speed
extern shared_data<uint8_t>*  p_reverse_direction_flag; // Reverse motor direction flag
extern shared_data<uint8_t>*  p_increase_power_flag;    // Increase motor power flag
extern shared_data<uint8_t>*  p_decrease_power_flag;    // Decrease motor power flag
extern shared_data<uint8_t>*  p_turn_left_flag;         // Decrease motor power flag
extern shared_data<uint8_t>*  p_turn_right_flag;        // Decrease motor power flag
extern shared_data<uint8_t>*  p_straight_flag;          // Decrease motor power flag
extern shared_data<double>*   p_sensor1;                // Pressure sensor 1 reading
extern shared_data<double>*   p_sensor2;                // Pressure sensor 2 reading
extern shared_data<double>*   p_sensor3;                // Pressure sensor 3 reading
extern shared_data<double>*   p_sensor4;                // Pressure sensor 4 reading
extern shared_data<double>*   p_sensor5;                // Pressure sensor 5 reading
extern shared_data<double>*   p_sensor6;                // Pressure sensor 6 reading
extern shared_data<double>*   p_sensor7;                // Pressure sensor 7 reading
extern shared_data<double>*   p_sensor8;                // Pressure sensor 8 reading
extern shared_data<double>*   p_sensor9;                // Proximity sensor 1 voltage reading
extern shared_data<double>*   p_sensor10;               // Proximity sensor 2 voltage reading

#endif // _SHARES_H_
```

```
#-------------------------------------------------------------------------------
# File:    Makefile for an AVR project - Wall-Climbing Robot
#          The makefile is the standard way to control the compilation and linking of
#          C/C++ files into an executable file. This makefile is also used to control
#          the downloading of the executable file to the target processor and the
#          generation of documentation for the project.
#
# Version:  04-11-2004 JRR Original file
#           06-19-2006 JRR Modified to use AVR-JTAG-ICE for debugging
#           11-21-2008 JRR Added memory locations and removed extras for bootloader
#           11-26-2008 JRR Cleaned up; changed method of choosing programming method
#           11-14-2009 JRR Added make support to put library files into subdirectory
#           09-28-2012 JRR Restructured to work with FreeRTOS subdirectory
#           12-05-2014 LBT Updated source files
#
# Relies   The avr-gcc compiler and avr-libc library
# on:      The avrdude downloader, if downloading through an ISP port
#          AVR-Insight or DDD and avarice, if debugging with the JTAG port
#          Doxygen, for automatic documentation generation
#
# Copyright 2006-2012 by JR Ridgely.  This makefile is intended for use in educational
# courses only, but its use is not restricted thereto. It is released under the terms
# of the Lesser GNU Public License with no warranty whatsoever, not even an implied
# warranty of merchantability or fitness for any particular purpose. Anyone who uses
# this file agrees to take all responsibility for any and all consequences of that use.
#-------------------------------------------------------------------------------

# The name of the program you're building, usually the file which contains main().
# The name without its extension (.c or .cpp or whatever) must be given here.
TARGET = robot_main

# A list of the source (.c, .cc, .cpp) files in the project, including $(TARGET). Files
# in library subdirectories do not go in this list; they're automatically in LIB_OBJS
SRC = $(TARGET).cpp task_user.cpp adc.cpp motor.cpp my_motor_task.cpp encoder.cpp
my_encoder_task.cpp task_sensor.cpp

# Clock frequency of the CPU, in Hz. This number should be an unsigned long integer.
# For example, 16 MHz would be represented as 16000000UL.
F_CPU = 16000000UL

# These codes are used to switch on debugging modes if they're being used. Several can
# be placed on the same line together to activate multiple debugging tricks at once.
# -DSERIAL_DEBUG       For general debugging through a serial device
# -DTRANSITION_TRACE   For printing state transition traces on a serial device
# -DTASK_PROFILE       For doing profiling, measurement of how long tasks take to run
# -DUSE_HEX_DUMPS      Include functions for printing hex-formatted memory dumps
OTHERS = -DSERIAL_DEBUG

# If the code -DTASK_SETUP_AND_LOOP is specified, ME405/FreeRTOS tasks classes will be
# required to provide methods setup() and loop(). Otherwise, they must only provide a
# a method called run() which is called just once by the scheduler.
OTHERS +=
```

```
# Other codes, used for turning on special features in a given project, can be put here
# -DSWOOP_BOARD        Tells the nRF24L01 radio driver to set up for the Swoop 1 board
# -DME405_BOARD_V05    Sets up radio driver for old ME405 board with 1 motor driver
# -DME405_BOARD_V06    Sets up radio driver for new ME405 board with 2 motor drivers
# -DME405_BREADBOARD   Sets up radio driver for ATmegaXX 40-pin on breadboard
# -DPOLYDAQ_BOARD      Sets up radio and other stuff for a PolyDAQ board
OTHERS += -DSWOOP_BOARD

# This define is used to choose the type of programmer from the following options:
# bsd        - Parallel port in-system (ISP) programmer using SPI interface on AVR
# jtagice    - Serial or USB interface JTAG-ICE mk I clone from ETT or Olimex
# bootloader - Resident program in the AVR which downloads through USB/serial port
# PROG = bsd
# PROG = jtagice
# PROG = bootloader
PROG = usbtiny

# These defines specify the ports to which the downloader device is connected.
# PPORT is for "bsd" on a parallel port, lpt1 on Windows or /dev/parport0 on Linux.
# JPORT is for "jtagice" on a serial port such as com1 or /dev/ttyS0, or usb-serial
#       such as com4 or /dev/ttyUSB1, or aliased serial port such as /dev/avrjtag
# BPORT is for "bootloader", the USB/serial port program downloader on the AVR
# The usbtiny programmer doesn't need a port specification; it has a USB identifier
PPORT = /dev/parport0
JPORT = /dev/ttyUSB1
BPORT = /dev/ttyUSB0

#-------------------------------------------------------------------------------
# This section specifies the type of CPU; uncomment one line for your processor. To add
# a new chip to the file, put its designation here and also set fuse bytes below.

# MCU = atmega128
MCU = atmega1281
# MCU = atmega32
# MCU = atmega324p
# MCU = atmega328p
# MCU = atmega644
# MCU = atmega644p
# MCU = atmega1284p

################################################################################
################ End of the stuff the user is expected to need to change ##############

# This is the name of the library file which will hold object code which has been
# compiled from all the source files in the library subdirectories
LIB_NAME = me405.a

# A list of directories in which source files (*.cpp, *.c) and headers (.h) for the
# library are kept
LIB_DIRS = lib/freertos lib/frtcpp lib/misc lib/comm lib/sensors lib/sd_card \
           lib/serial

#-------------------------------------------------------------------------------
```

```
# In this section, default settings for fuse bytes are given for each processor which
# this makefile supports. New chip specifications can be added to this file as needed.

# ATmega128 set up for ME405 board with JTAG enabled
ifeq ($(MCU), atmega128)
  EFUSE = 0xFF
  HFUSE = 0x11
  LFUSE = 0xEF
# ATmega128 set up for Wall-Climbing Robot board with JTAG disabled
else ifeq ($(MCU), atmega1281)
  EFUSE = 0xFF
  HFUSE = 0xD1
  LFUSE = 0xEF
# ATmega32 configured for Swoop sensor board with JTAG disabled to save power
else ifeq ($(MCU), atmega32)
  EFUSE =
  HFUSE = 0x99
  LFUSE = 0xEF
# ATmega324P configured for Swoop sensor board with JTAG disabled
# Standard fuses FF19EF, bootloader fuses FFC8EF, low power fuses FF11EF
else ifeq ($(MCU), atmega324p)
  EFUSE = 0xFF
  HFUSE = 0x11
  LFUSE = 0xEF
# ATmega328P configured for Super Ministrone board
# Standard fuses FF19EF, bootloader fuses FFC8EF, low power fuses FF11EF
else ifeq ($(MCU), atmega328p)
  EFUSE = 0x07
  HFUSE = 0xD1
  LFUSE = 0xEF
# ATmega644 (note: the 644P needs a different MCU)
else ifeq ($(MCU), atmega644)
  EFUSE = 0xFF
  HFUSE = 0x11
  LFUSE = 0xEF
# ATmega644P configured for Swoop-2 sensor board
# Standard fuses FF19EF, bootloader fuses FFC8EF, low power fuses FF11EF, JTAG OFF!
else ifeq ($(MCU), atmega644p)
  EFUSE = 0xFF
  HFUSE = 0xD1
  LFUSE = 0xEF
# ATmega1284P configured for Swoop sensor board
# Standard fuses FF19EF, bootloader fuses FFC8EF, low power fuses FF11EF
else ifeq ($(MCU), atmega1284p)
  EFUSE = 0xFF
  HFUSE = 0xD1
  LFUSE = 0xEF
endif

#---------------------------------------------------------------------------------
# Tell the compiler how hard to try to optimize the code. Optimization levels are:
# -O0  Don't try to optimize anything (even leaves empty delay loops in)
# -O1  Some optimizations; code usually smaller and faster than O0
```

```
# -O2  Pretty high level of optimization; often good compromise of speed and size
# -O3  Tries really hard to make code run fast, even if code size gets pretty big
# -Os  Tries to make code size small. Sometimes -O1 makes it smaller, though(?!)
OPTIM = -O2

# Warnings which need to be given
C_WARNINGS = -Wall -Wextra -Wshadow -Wpointer-arith -Wbad-function-cast -Wcast-align \
             -Wsign-compare -Wstrict-prototypes -Wmissing-prototypes -Wunused \
             -Wmissing-declarations -Waggregate-return

CPP_WARNINGS = -Wall -Wextra -Wshadow -Wpointer-arith -Wcast-align -Wsign-compare \
               -Wmissing-declarations -Wunused

C_FLAGS = -D GCC_MEGA_AVR -D F_CPU=$(F_CPU) -D _GNU_SOURCE \
          -fsigned-char -funsigned-bitfields -fpack-struct -fshort-enums \
          -std=gnu99 -g $(OPTIM) -mmcu=$(MCU) $(OTHERS) $(C_WARNINGS) \
          $(patsubst %,-I%,$(LIB_DIRS))

CPP_FLAGS = -D GCC_MEGA_AVR -D F_CPU=$(F_CPU) -D _GNU_SOURCE \
            -fsigned-char -funsigned-bitfields -fshort-enums \
            -g $(OPTIM) -mmcu=$(MCU) $(OTHERS) $(CPP_WARNINGS) \
            $(patsubst %,-I%,$(LIB_DIRS))

# This section makes a list of object files from the source files in the SRC list,
# separating the C++ source files, the C source files, and assembly source files
OBJS = $(patsubst %.cpp, %.o, $(filter %.cpp, $(SRC))) \
       $(patsubst %.c, %.o, $(filter %.c, $(SRC))) \
       $(ASRC:.S=.o)

# This section makes a list of object files from the source files in subdirectories
# in the LIB_DIRS list, separating the C++, C, and assembly source files
LIB_SRC =  $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.cpp)) \
           $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.cc)) \
           $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.c)) \
           $(foreach A_DIR, $(LIB_DIRS), $(wildcard $(A_DIR)/*.S))

LIB_OBJS = $(patsubst %.cpp, %.o, $(filter %.cpp, $(LIB_SRC))) \
           $(patsubst %.cc, %.o, $(filter %.cc, $(LIB_SRC))) \
           $(patsubst %.c, %.o, $(filter %.c, $(LIB_SRC))) \
           $(LIB_ASRC:.S=.o)

#---------------------------------------------------------------------------------------
# Inference rules show how to process each kind of file.

# How to compile a .c file into a .o file
.c.o:
    @echo $<
    @avr-gcc -c $(C_FLAGS) $< -o $@

# How to compile a .cc file into a .o file
.cc.o:
    @echo $<
    @avr-gcc -c $(CPP_FLAGS) $< -o $@
```

```
# How to compile a .cpp file into a .o file
.cpp.o:
    @echo $<
    @avr-gcc -c $(CPP_FLAGS) $< -o $@

#-------------------------------------------------------------------------------
# Make the main target of this project.  This target is invoked when the user types
# 'make' as opposed to 'make <target>.'  This must be the first target in Makefile.

all:  $(TARGET).hex

#-------------------------------------------------------------------------------
# This rule creates a .hex format downloadable file. A raw binary file which can be
# used by some bootloaders can be created; a listing file is also created.

$(TARGET).hex:  $(TARGET).elf
    @avr-objdump -h -S $(TARGET).elf > $(TARGET).lst
    @avr-objcopy -j .text -j .data -O ihex $(TARGET).elf $(TARGET).hex
    @avr-size $(TARGET).elf

#-------------------------------------------------------------------------------
# This rule controls the linking of the target program from object files. The target
# is saved as an ELF debuggable binary.

$(TARGET).elf:  library $(OBJS)
    avr-gcc $(OBJS) $(LIB_NAME) -g -mmcu=$(MCU) -o $(TARGET).elf

#-------------------------------------------------------------------------------
# This is a dummy target that doesn't do anything. It's included because the author
# belongs to a faculty labor union and has been instilled with reverence for laziness.

nothing:

#-------------------------------------------------------------------------------
# 'make install' will make the project, then download the program using whichever
# method has been selected -- ISP cable, JTAG-ICE module, or USB/serial bootloader

install:  $(TARGET).hex
  ifeq ($(PROG), bsd)
    avrdude -p $(MCU) -P $(PPORT) -c bsd -V -E noreset -Uflash:w:$(TARGET).hex
  else ifeq ($(PROG), jtagice)
    avarice -e -p -f $(TARGET).elf -j $(JPORT)
  else ifeq ($(PROG), bootloader)
    @echo "ERROR: No bootloader set up for this Makefile"
  else ifeq ($(PROG), usbtiny)
    avrdude -p $(MCU) -c usbtiny -V -B 1 -Uflash:w:$(TARGET).hex
  else
    @echo "ERROR: No programmer" $(PROG) "in the Makefile"
  endif

#-------------------------------------------------------------------------------
# 'make fuses' will set up the processor's fuse bits in a "standard" mode. Standard is
```

```
# a setup in which there is no bootloader but the ISP and JTAG interfaces are enabled.

fuses: nothing
  ifeq ($(PROG), bsd)
    avrdude -p $(MCU) -P $(PPORT) -c $(PROG) -V -E noreset -Ulfuse:w:$(LFUSE):m
    avrdude -p $(MCU) -P $(PPORT) -c $(PROG) -V -E noreset -Uhfuse:w:$(HFUSE):m
    avrdude -p $(MCU) -P $(PPORT) -c $(PROG) -V -E noreset -Uefuse:w:$(EFUSE):m
  else ifeq ($(PROG), jtagice)
    @echo "ERROR: Fuse byte programming not set up for JTAG in this Makefile"
  else ifeq ($(PROG), usbtiny)
    avrdude -p $(MCU) -c usbtiny -q -V -E noreset -Ulfuse:w:$(LFUSE):m
    avrdude -p $(MCU) -c usbtiny -q -V -E noreset -Uhfuse:w:$(HFUSE):m
    avrdude -p $(MCU) -c usbtiny -q -V -E noreset -Uefuse:w:$(EFUSE):m
  else
    @echo "ERROR: Only bsd or USBtiny set to program fuse bytes in this Makefile"
  endif

#-------------------------------------------------------------------------------------
# 'make readfuses' will see what the fuses currently hold

readfuses: nothing
  ifeq ($(PROG), bsd)
    @echo "ERROR: Not yet programmed to read fuses with bsd/ISP cable"
  else ifeq ($(PROG), jtagice)
    @avarice -e -j $(JPORT) --read-fuses
  else ifeq ($(PROG), bootloader)
    @echo "ERROR: Not yet programmed to read fuses via bootloader"
  else
    @echo "ERROR: No known device specified to read fuses"
  endif

#-------------------------------------------------------------------------------
# 'make reset' will read a byte of lock bits, ignore it, and reset the chip

reset:
  ifeq ($(PROG), bsd)
    avrdude -c bsd -p $(MCU) -P $(PPORT) -c $(PROG) -V -E noreset \
        -Ulfuse:r:/dev/null:r
  else ifeq ($(PROG), usbtiny)
    avrdude -p $(MCU) -c usbtiny -q -V -Ulfuse:r:/dev/null:r
  else
    @echo "ERROR: make reset only works with parallel ISP cable"
  endif

#-------------------------------------------------------------------------------------
# 'make doc' will use Doxygen to create documentation for the project. 'make libdoc'
# will do the same for the subdirectories which include ME405 library files.

.PHONY: doc libdoc
doc:
    @doxygen doxygen.conf

libdoc:
```

```
    @doxygen doxy_lib.conf

#------------------------------------------------------------------------------
# 'make clean' will erase the compiled files, listing files, etc. so you can restart
# the building process from a clean slate. It's also useful before committing files to
# a CVS or SVN or Git repository.

clean:
    @echo -n Cleaning compiled files and documentation...
    @rm -f $(LIB_NAME) *.o *.hex *.lst *.elf *~
    @for subdir in $(LIB_DIRS); do \
        rm -f $$subdir/*.o; \
        rm -f $$subdir/*.lst; \
        rm -f $$subdir/*~; \
    done
    @echo done.

#------------------------------------------------------------------------------
# 'make library' will build the library file, using an automatically generated list of
# all the C, C++, and assembly source files in the library directories in LIB_DIRS

library: $(LIB_OBJS)
    @avr-ar -r $(LIB_NAME) $(LIB_OBJS)

#------------------------------------------------------------------------------
# 'make term' will run a PuTTY terminal using the settings that are most commonly used
# to talk to an AVR microcontroller connected through a USB-serial port. One must have
# installed PuTTY on the local computer and configured the default settings, of course

term:
    @putty -load "Default Settings" &

#------------------------------------------------------------------------------
# 'make help' will show a list of things this makefile can do

help:
    @echo 'make          - Build program file ready to download'
    @echo 'make install  - Build program and download with parallel ISP cable'
    @echo 'make reset    - Reset processor with parallel cable RESET line'
    @echo 'make doc      - Generate documentation with Doxygen'
    @echo 'make clean    - Remove compiled files from all directories'
    @echo ' '
    @echo 'Notes: 1. Other less commonly used targets are in the Makefile'
    @echo '       2. You can combine targets, as in "make clean all"'
```

```
//*****************************************************************************
/** \file adc.h
 *    This file contains a very simple A/D converter driver for 8 analog differential
 *    vacuum pressure sensors and 2 proximity sensors. Six of the sensors are connected
 *    to a multiplexer, which is enabled using general output ports to obtain those
 *    readings.
 *
 *  Revisions:
 *    \li 01-15-2008 JRR Original (somewhat useful) file
 *    \li 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
 *    \li 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
 *    \li 12-04-2014 LBT Updated comments
 *    \li 05-13-2015 LBT Updated formatting and comments
 *    \li 05-30-2015 LBT Removed unnecessary method to print ADC register
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#ifndef _AVR_ADC_H_              // This define prevents this .H file from
#define _AVR_ADC_H_              // being included multiple times in a .CPP file

#include "emstream.h"            // Header for serial ports and devices
#include "FreeRTOS.h"            // Header for the FreeRTOS RTOS
#include "task.h"                // Header for FreeRTOS task functions
#include "queue.h"               // Header for FreeRTOS queues
#include "semphr.h"              // Header for FreeRTOS semaphores
#include <avr/interrupt.h>       // Header for interrupts

//*****************************************************************************
/** \brief This class should run the A/D converter for 10 analog sensors.
 *  \details This class runs the A/D converter for 10 analog sensors.
 */

class adc
{
    private:
        // No private variables or methods in this class.

    protected:
        // The ADC class uses this pointer to the serial port to communicate.
```

```
        emstream* ptr_to_serial;

    public:
        // The constructor sets up the A/D converter for use.
        adc (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method reads one channel once, returning the result as an unsigned
        // integer; it should be called from within a normal task, not an ISR.
        uint16_t read_once (uint8_t);

        // This method reads the A/D a designated amount of times and returns the average.
        // Doing so implements a crude sort of low-pass filtering that can help reduce noise.
        uint16_t read_oversampled (uint8_t, uint8_t);

}; // end of class adc

#endif // _AVR_ADC_H_
```

```
//*****************************************************************************
/** \file adc.cpp
 *    This file contains a very simple A/D converter driver and controls the analog data
 *    acquisition for 8 differential vacuum pressure sensors and 2 proximity sensors.
 *
 *  Revisions:
 *    \li 01-15-2008 JRR Original (somewhat useful) file
 *    \li 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
 *    \li 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
 *    \li 12-04-2014 LBT Updated ADC channels used and comments
 *    \li 05-13-2015 LBT Updated formatting and comments
 *    \li 05-30-2015 LBT Removed unnecessary method to print ADC register
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */

//*****************************************************************************
#include <stdlib.h>                         // Include standard library header files
#include <avr/io.h>                         // Include Port I/O for SFR's
#include "rs232int.h"                       // Include header for serial port class
#include "adc.h"                            // Include header for the A/D class

//*****************************************************************************
/** \brief   This constructor sets up and initializes the A/D converter.
 *  \details The A/D is made ready so that when a  method such as read_once() is
 *           called, correct A/D conversions can be performed. The constructor
 *           accomplishes 4 tasks. First, it turns on the ADC and sets the clock
 *           prescaler. Secondly, it selects the reference voltage for ADC measurements
 *           and then sets the default channel as 0.
 *  @param   a_name - A character string which will be the name of this task.
 *  @param   a_priority - The priority at which this task will initially run (default: 2).
 *  @param   a_stack_size - The size of this task's stack in bytes.
 *  @param   p_ser_dev - Pointer to the Bluetooth serial device which can be used by this
 *           task to communicate. (default: bluetooth_ser_port).
 *  @return  No return values
 */

adc::adc (const char*, unsigned portBASE_TYPE, size_t, emstream* p_ser_dev)

{
    ptr_to_serial = p_ser_dev;                  // Create name for serial pointer
```

```
    ADCSRA = (1<<ADEN)|                         // Turn on ADC, prescaler = 32 (page 292)
             (1<<ADPS2)|
             (1<<ADPS0);
    ADMUX  = (1<<REFS0);                        // Selected AVCC external cap @ reference pin
    ADCSRB = (1<<ACME);                         // Analog comparator multiplexer enable
}

//*****************************************************************************
/** \brief   This method takes one A/D reading from the given channel and returns it.
 *  \details This function will take one sample from the onboard ADC and return the result
 *           as a 16 bit number. Note that the answer only uses the lower 10 bits. A loop
 *           is used to make sure that the ADC conversion cannot last forever.
 *  @param   ch - The A/D channel which is being read (must be from 0 to 7).
 *  @return  conversion_complete - The result of the A/D conversion.
 */

uint16_t adc::read_once (uint8_t ch)
{
    uint16_t conversion_complete = 0;                   // ADC result variable
    uint8_t low = 0;                                    // Variable for low bit
    uint8_t  high = 0;                                  // Variable for high bit
    int loop = 0;                                       // Loop timer
    ADMUX &= 0xF8;                                      // Reset ADMUX register
    ADMUX  |= (ch & 0x07);                              // Determine which ADC channel is being
    sampled
    ADCSRA |= (1 << ADSC);                              // Start conversion
    while(loop < 10001)                                 // Wait for conversion to complete
    {
      if(ADCSRA == 0x95)                                // Check to see if conversion complete
      {
        low = ADCL;                                     // Read low ADC data bits
        high = ADCH;                                    // Read high ADC data bits
        conversion_complete |= (uint16_t) (high << 8);  // Shift high bits into 16 bit number
        conversion_complete |= (uint16_t) low;          // Add low bits to 16 bit number
        loop = 10001;                                   // Set counter to exit loop
      }
      loop++;                                           // Increment counter if conversion not
      done
    }
    return (conversion_complete);                       // Return converted number
}

//*****************************************************************************
/** \brief   This method averages the ADC measurements over a specified number of samples.
 *  \details The user enters the desired number of samples where the method is called.
 *           The ADC measurement is then averaged over the desired number of samples.
 *  @param   channel - The ADC channel currently being read and monitored.
 *  @param   samples - The number of samples the ADC will average over (max = 64).
 *  @return  output - Average ADC measurement over the number of samples specified.
 */

uint16_t adc::read_oversampled (uint8_t channel, uint8_t samples)
{
```

```
    uint16_t answer = 0;                    // Storage space for ADC result
    uint16_t output = 0;                    // Storage space for averaged result
    if(samples < 64)                        // Check to see if sample will not overflow
    {
      for(uint8_t i = 0; i<samples; i++)    // Loop for number of samples
      {
        answer += read_once(channel);       // Add the new sample to the previous one
      }
      output = answer/samples;              // Find the average output
    }
    else                                    // Sample range to large
    {
      output = 0;                           // Return the result as zero
    }
    return (output);                        // Return the result to the location
}                                           // from which the method was called
```

```
//*******************************************************************************
/** \file task_sensor.h
 *    This file contains the header for a task class that controls the pressure sensor data.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task
 *    \li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    \li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    \li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    \li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *    \li 12-05-2014 LBT Updated file for sensor functionality
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*******************************************************************************

#ifndef _TASK_SENSOR_H_              // This define prevents this .h file from being
#define _TASK_SENSOR_H_              // included multiple times in a .cpp file

#include <stdlib.h>                          // Prototype declarations for I/O functions
#include <avr/io.h>                          // Header for special function registers
#include "FreeRTOS.h"                        // Primary header for FreeRTOS
#include "task.h"                            // Header for FreeRTOS task functions
#include "queue.h"                           // FreeRTOS inter-task communication queues
#include "frt_task.h"                        // ME405/507 base task class
#include "time_stamp.h"                      // Class to implement a microsecond timer
#include "frt_queue.h"                       // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                 // Header for thread-safe shared data
#include "rs232int.h"                        // ME405/507 library for serial comm.
#include "adc.h"                             // Header for A/D converter driver class

//*******************************************************************************
/** \brief This class outputs sensor values using an analog input from the A/D converter.
 *  \details The A/D converter is run using a driver in files \c adc.h and \c adc.cpp.
 */

class task_sensor : public frt_task
{
private:
    // The task_sensor class uses this pointer to the serial port to say hello.
```

```
    emstream* ptr_to_serial;

protected:
    // No protected variables or methods for this class.

public:
    // This constructor creates a generic task of which many copies can be made.
    task_sensor (const char*, unsigned portBASE_TYPE, size_t, emstream*);

    // This method is called by the RTOS once to run the task loop for ever and ever.
    void run (void);
};

#endif // _TASK_SENSOR_H_
```

```
//*****************************************************************************
/** \file task_sensor.cpp
 *    This file contains the code for a task class which controls the pressure sensor data.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task
 *    \li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    \li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    \li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    \li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *    \li 12-01-2014 LBT Converted to a pressure sensor task that outputs ADC readings
 *                       through a bluetooth module
 *    \li 05-29-2015 LBT Added shared variables and code to output proximity sensor values
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#include "frt_task.h"                           // Header of wrapper for FreeRTOS tasks
#include "frt_text_queue.h"                     // Wrapper for FreeRTOS character queues
#include "frt_queue.h"                          // Header of wrapper for FreeRTOS queues
#include "task_sensor.h"                        // Header for this task
#include "shares.h"                             // Shared inter-task communications
#include "adc.h"                                // Include ADC task functions
#include "math.h"

//*****************************************************************************
/** \brief   This constructor creates the sensor task.
 *  \details This constructor creates a task which determines vacuum pressure readings using
 *           input from an A/D converter. The main job of this constructor is to call the
 *           constructor of parent class (\c frt_task ); the parent's constructor the work.
 *  @param   a_name - A character string which will be the name of this task
 *  @param   a_priority - The priority at which this task will initially run (default: 0).
 *  @param   a_stack_size - The size of this task's stack in bytes. (default:
 configMINIMAL_STACK_SIZE).
 *  @param   p_ser_dev - Pointer to a serial device (port, radio, SD card, etc.) which can
 *           be used by this task to communicate. (default: NULL).
 *  @return  No return values.
 */
```

```
task_sensor::task_sensor (const char* a_name, unsigned portBASE_TYPE a_priority, size_t
a_stack_size, emstream* p_ser_dev)
            :frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
    ptr_to_serial = p_ser_dev;
    // Nothing much is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one.
}

//***************************************************************************
/** \brief   This method is called once by the RTOS scheduler to run this task.
 *  \details Each time around the for (;;) loop, it reads the A/D converter and uses the
 *           result to determine the sensor reading.
 *  @param   No parameters.
 *  @return  No return values.
 */

void task_sensor::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling.
    portTickType previousTicks = xTaskGetTickCount ();

    // Create an analog to digital converter driver object and a variable in which to
    // store its output. The variable p_my_adc only exists within this run() method,
    // so the A/D converter cannot be used from any other function or method.
    adc* p_my_adc = new adc("ADC", task_priority (2), 280, p_serial);
    double a2d_reading = 0;
    double voltage = 0;
    double voltage_p = 0;
    double pressure = 0;
    double sensor9 = 0;
    double sensor10 = 0;
    double proximity = 0;
    DDRA = 0x0F;                                    // Set multiplexer select pins as output

    // This is the task loop for the sensor control task. This loop runs until the
    // power is turned off or something equally dramatic occurs.
    for (;;)
    {
        // Read the A/D converter. (10bit)
        // Read the ADC, user must enter channel number and samples amount
        // can measure channels 0-7 and up to 64 samples.

        // Sensor 1
        a2d_reading = p_my_adc-> read_oversampled (0,32);                   // Get ADC reading
        voltage = 5 * a2d_reading / 1023;                                   // Calculate voltage
        pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)-0.5);  // Calculate pressure
        using sensor transfer function
        p_sensor1 -> put(pressure);                                         // Put sensor
        reading in global variable

        // Sensor 2
        a2d_reading = p_my_adc-> read_oversampled (1,32);
```

```
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)-0.7);
p_sensor2 -> put(pressure);

// Sensor 3
a2d_reading = p_my_adc-> read_oversampled (2,32);
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)+0.2);
p_sensor3 -> put(pressure);

// Sensor 4
a2d_reading = p_my_adc-> read_oversampled (3,32);
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)+0.5);
p_sensor4 -> put(pressure);

// Sensor 5
PORTA = 0x00;
a2d_reading = p_my_adc-> read_oversampled (7,32);
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)-0.3);
p_sensor5 -> put(pressure);

// Sensor 6
PORTA = 0x01;
a2d_reading = p_my_adc-> read_oversampled (7,32);
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009));
p_sensor6 -> put(pressure);

// Sensor 7
PORTA = 0x02;
a2d_reading = p_my_adc-> read_oversampled (7,32);
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)+0.7);
p_sensor7 -> put(pressure);

// Sensor 8
PORTA = 0x03;
a2d_reading = p_my_adc-> read_oversampled (7,32);
voltage = 5 * a2d_reading / 1023;
pressure = 101.325 - ((voltage / 0.045) - (0.04 / 0.009)+0.5);
p_sensor8 -> put(pressure);

// Proximity #1
PORTA = 0x0C;
a2d_reading = p_my_adc-> read_oversampled (7,32);
voltage_p = 5 * a2d_reading / 1023;
sensor9 = voltage_p;
p_sensor9 -> put(voltage_p);

// Proximity #2
PORTA = 0x09;
```

```
        a2d_reading = p_my_adc-> read_oversampled (7,32);
        voltage_p = 5 * a2d_reading / 1023;
        sensor10 = voltage_p;
        p_sensor10 -> put(voltage_p);

        if(sensor9 > sensor10)
        {
            proximity = 0.0346*(1/sensor9)*(1/sensor9) + 0.251*(1/sensor9) - 0.0121;
        }
        if(sensor10 > sensor9)
        {
            proximity = 0.0346*(1/sensor10)*(1/sensor10) + 0.251*(1/sensor10) - 0.0121;
        }

        p_proximity -> put(proximity);

        // Increment the run counter. This counter belongs to the parent class and can
        // be printed out for debugging purposes.
        runs++;

        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times.
        delay_from_to (previousTicks, configMS_TO_TICKS (100));
    }
}
```

```
//*********************************************************************************
/** \file my_encoder_task.h
 *    This file contains the header file for a task which reads and deciphers the encoder
 *    state changes. User input from the GUI controls the clearing of the encoder position count.
 *
 *  Revisions:
 *    \li 01-28-2014 LBT Created file
 *    \li 12-01-2014 LBT Added two encoder functionality
 *    \li 05-30-2015 LBT Updated comments and formatting
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*********************************************************************************

#ifndef _MY_ETASK_H_                // This define prevents this .h file from being
#define _MY_ETASK_H_                // included multiple times in a .cpp file

#include <stdlib.h>                 // Prototype declarations for I/O functions
#include <avr/io.h>                 // Header for special function registers
#include "FreeRTOS.h"               // Primary header for FreeRTOS
#include "task.h"                   // Header for FreeRTOS task functions
#include "queue.h"                  // FreeRTOS inter-task communication queues
#include "frt_task.h"               // ME405/507 base task class
#include "time_stamp.h"             // Class to implement a microsecond timer
#include "frt_queue.h"              // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"        // Header for thread-safe shared data
#include "rs232int.h"               // ME405/507 library for serial comm.
#include "adc.h"                    // Header for A/D converter driver class
#include "motor.h"                  // Needed to access the motor class
#include <avr/interrupt.h>          // Needed to utilize interrupts
#include "encoder.h"                // Needed to access encoder class
#include "shares.h"                 // Needed to access shared data variables

//-------------------------------------------------------------------------------------
/** \brief   This task reads and deciphers motor encoders.
 *  \details The encoders are read using a driver and interrupts in files encoder.h and
 encoder.cpp.
 *           This code sets up and reads the encoders.
 */

class my_encoder_task : public frt_task
```

```
{
private:
    // Create serial connection for debugging.
    emstream* ptr_to_serial;

protected:
    // No protected variables or methods for this class.

public:
    // This constructor creates a generic task of which many copies can be made.
    my_encoder_task (const char*, unsigned portBASE_TYPE, size_t, emstream*);

    // This method is called by the RTOS once to run the task loop for ever and ever.
    void run (void);
};

#endif // _MY_ETASK_H_
```

```
//******************************************************************************
/** \file my_encoder_task.cpp
 *    This file contains code for a task which initializes the encoders and runs a continuous
 loop
 *    to check if encoder positions need to be displayed or cleared by checking for
 *    flags that are set from user input obtained in task_user.
 *
 *  Revisions:
 *    \li 01-28-2014 LBT Created file
 *    \li 12-01-2014 LBT Added two encoder functionality and motor speed calculation
 *    \li 12-05-2014 LBT Updated comments
 *    \li 05-30-2015 LBT Removed errorcount code and updated encoder methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//******************************************************************************

#include "frt_text_queue.h"             // Header for text queue class
#include "my_encoder_task.h"            // Header for this task
#include "rs232int.h"                   // Include header for serial port class
#include <avr/interrupt.h>              // Needed to utilize interrupts
#include "shares.h"                     // Shared inter-task communications
#include "math.h"

//--------------------------------------------------------------------------------
/** \brief   This constructor creates the my_encoder_task.
 *  \details This constructor creates a task which reads the encoders through the use of
 interrupts
 *           and user input.
 *  @param   a_name A character string which will be the name of this task.
 *  @param   a_priority The priority at which this task will initially run (default: 2).
 *  @param   a_stack_size The size of this task's stack in bytes. (default:
 configMINIMAL_STACK_SIZE).
 *  @param   p_ser_dev Pointer to a Bluetooth serial device which can be used by this task
 *           to communicate. (default: bluetooth_ser_port).
 *  @return  No return values.
 */

my_encoder_task::my_encoder_task (const char* a_name, unsigned portBASE_TYPE a_priority, size_t
a_stack_size, emstream* p_ser_dev)
                :frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
```

```
{
  ptr_to_serial = p_ser_dev;            // Create name for the debugging serial pointer
  // Nothing much is done in the body of this constructor. All the work is done in the
  // call to the frt_task constructor on the line just above this one.
}

//-------------------------------------------------------------------------------------
/** \brief   This method runs the encoder functionality.
 *  \details This method is called once by the RTOS scheduler. Each time around the for (;;)
 *           loop, it checks the shared data variables used as flags and calls the appropriate
 method.
 *  @param   No parameters.
 *  @return  No return values.
 */

void my_encoder_task::run (void)
{
  portTickType previousTicks = xTaskGetTickCount ();     // Obtain tick count from MCU
  encoder *encoder1 = new encoder();                     // Create one encoder instance of class
  encoder

  uint8_t zero_position = 0;                             // Initialize zero position flag
  int32_t countA = 0;                        // Create countA variable
  int32_t countB = 0;                        // Create countB variable
  double old_positionA = 0;                              // Initialize motor A and B position
  variables
  double current_positionA = 0;
  double position_changeA = 0;
  double old_position = 0;                               // Initialize motor A and B position
  variables
  double current_position = 0;
  double position_change = 0;
  double old_positionB = 0;
  double current_positionB = 0;
  double position_changeB = 0;
  double current_speedA = 0;
  double current_speedB = 0;
  double current_speed = 0;
  double speed = 0;
  //double thetaA = 0;
  //double thetaB = 0;
  int movement = 0;
  int i = 1;

  for (;;)                                               // This is the task loop for the
  encoder task. This loop
  {                                                      // runs until the power to the
  board is turned off.
    zero_position = p_zeropositionflag -> get();         // Load current flag statuses into
    local variables
    countA = p_countA -> get();
    countB = p_countB -> get();
    //movement = p_movement -> get();
```

```
    //thetaA = p_thetaA -> get();
    //thetaB = p_thetaB -> get();

    if(i == 2)
    {
    encoder1 -> currentposition();                          // Update current positions

/*     old_positionA = current_positionA;                       // Set current count position
as old position
    current_positionA = p_positionA -> get();               // Get updated current position
    position_changeA = current_positionA - old_positionA;   // Calculate ticks since last run
    through loop
    if(position_changeA < 0)
    {
        position_changeA = -position_changeA;               // Find magnitude of ticks (always
        positive speed)
    }

    old_positionB = current_positionB;                      // Set current count position as
    old position
    current_positionB = p_positionB -> get();               // Get updated current position
    position_changeB = current_positionB - old_positionB;   // Calculate ticks since last run
    through loop
    if(position_changeB < 0)
    {
        position_changeB = -position_changeB;               // Find magnitude of ticks (always
        positive speed)
    }    */

    old_position = current_position;                        // Set current count position as
    old position
    current_position = p_position -> get();             // Get updated current position
    position_change = current_position - old_position;  // Calculate ticks since last run
    through loop
    if(position_change < 0)
    {
        position_change = -position_change;             // Find magnitude of ticks (always
        positive speed)
    }

     current_speed = position_change/0.1;

    /* current_speedA = position_changeA/0.1;              // Calculate current motor rotation
    speeds
    current_speedB = position_changeB/0.1;
    p_motor_speedA -> put(current_speedA);                  // Update global speed variables
    p_motor_speedB -> put(current_speedB);
    speed = (current_speedA + current_speedB)/2;            // Average two speeds together to
    obtain robot speed */
    p_speed -> put(current_speed);

    i = 0;
    }
```

```
    i++;

    current_positionA = p_y_position -> get();
    current_positionB = p_x_position -> get();
    //DBG (ptr_to_serial, current_speed << endl);
    //DBG (ptr_to_serial, current_positionB << "  " << current_positionA << endl);

    if(zero_position == 1)                                  // If user input is to zero
    {                                                       // encoder position count
      encoder1 -> zeroposition();                           // go to the zero position
      old_positionA = 0;                                    // method in encoder.cpp
      old_positionB = 0;                                    // Clear all position variables in
      this loop
    }

    // Increment the run counter. This counter belongs to the parent class and can
    // be printed out for debugging purposes.
    runs++;

    // This is a method we use to cause a task to make one run through its task
    // loop every N milliseconds and let other tasks run at other times.
    delay_from_to (previousTicks, configMS_TO_TICKS (100));
  }
}
```

```
//=============================================================================================
/** \file encoder.h
 *    This file contains definitions for the encoder driver.
 *
 *  Revisions:
 *    \li 02-10-2014 LBT Original file designed
 *    \li 12-01-2014 LBT Updated comments and added private variables heading
 *    \li 05-30-2015 LBT Removed errorcount code and updated encoder methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//=============================================================================================
#ifndef _AVR_ENCODER_H_              // This define prevents this .H file from being
#define _AVR_ENCODER_H_              // included multiple times in a .CPP file

#include <stdlib.h>                  // Include standard library header files
#include <avr/io.h>                  // Port I/O for SFR's
#include "emstream.h"                // Header for serial ports and devices
#include "FreeRTOS.h"                // Header for the FreeRTOS RTOS
#include "task.h"                    // Header for FreeRTOS task functions
#include "queue.h"                   // Header for FreeRTOS queues
#include "semphr.h"                  // Header for FreeRTOS semaphores

//---------------------------------------------------------------------------------------------
/** \brief   This class should run the encoder driver for the motor.
 *  \details This class enables the encoder driver.
 */

class encoder
{
    private:
        // No private variables or methods in this class.

    protected:
        // The encoder class uses this pointer to the serial port to communicate.
        emstream* ptr_to_serial;

    public:
        // Constructor for encoder class.
        encoder (void);
```

```
        // Set up for the following methods.
        void currentposition(void);
        void zeroposition(void);

}; // end of class encoder

#endif // _AVR_ENCODER_H_
```

```
//*****************************************************************************
/** \file encoder.cpp
 *    This file contains code for a task which controls the two hall effect gear tooth
 *    encoders.
 *
 Revisions:
 *    \li 02-10-2014 LBT Original file designed
 *    \li 11-29-2014 LBT Added second encoder code
 *    \li 05-13-2015 LBT Removed code for quadrature encoder analysis and updated to
 *                       single interrupt analysis for each motor
 *    \li 05-30-2015 LBT Removed errorcount code and updated encoder methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */

//*****************************************************************************
#include "frt_text_queue.h"             // Header for text queue class
#include "encoder.h"                    // Header for this task
#include "rs232int.h"                   // ME405/507 library for serial comm.
#include <avr/interrupt.h>              // Needed to utilize interrupts
#include "my_encoder_task.h"            // Needed to use encoder task functions
#include "shares.h"                     // Shared inter-task communications
#include "math.h"

  double theta = 0;
  double thetaA = 0;
  double thetaB = 0;
  uint8_t mode = 1;
  //double positionA = 0;                     // Create positionA variable
  //double positionB = 0;                     // Create positionB variable
  double position = 0;                    // Create positionB variable
  double old_x_position = 0;                    // Create positionA variable
  double old_y_position = 0;
  double x_position = 0;                    // Create positionA variable
  double y_position = 0;
  int movement = 0;
  int32_t countA = 0;                   // Create countA variable
  int32_t countB = 0;                   // Create countB variable
  int32_t count = 0;                    // Create countA variable
  double theta_max = 0;
//*****************************************************************************
```

```
/** \brief   This constructor creates the encoder task.
 *  \details This constructor creates a task which initializes the encoder and interrupts.
 *  @param   p_serial_port - Pointer to a Bluetooth serial device which can be used by
 *           this task to communicate (default: bluetooth_ser_port).
 *  @return  No return values.
 */

encoder::encoder (void) // Encoder interrupt setup
{
  DDRE  &=  ~(1<<PE7);                    // Clear encoder ports
  DDRD  &=  ~(1<<PD1);
  EIMSK &=  ~(0xFF);                      // Clear mask register
  EICRA =    (1<<ISC10);                  // Detect rising edges
  EICRA &=  ~(1<<ISC11);
  EICRB =    (1<<ISC70);
  EICRB &=  ~(1<<ISC71);
  EIMSK =    (1<<INT7)|                   // Set active interrupts
             (1<<INT1);
}

//*****************************************************************************
/** \brief   This method displays the current position of the robot relative to the
 *           current zero position.
 *  \details This method calculates the current positions of the robot and saves them.
 *  @param   p_serial_port - Takes in a pointer to the serial port for output.
 *  @return  No return values.
 */

void encoder::currentposition(void)         // Outputs linear travel position
{
  theta_max = p_theta_max -> get();
  mode = p_modeA -> get();
  movement = p_movement -> get();
  old_x_position = p_x_position -> get();
  old_y_position = p_y_position -> get();
  countA = p_countA -> get();               // Get current value of countA
  countB = p_countB -> get();               // Get current value of countB
  //count = (countA + countB)/2;
  count = countB;
    // Inches (using resolution of gear)
  //positionB = countB*0.26016;             // Inches (using resolution of gear)
  if((movement == 0)  && (mode == 0))
  {
    position = count*0.26016;
    p_position -> put(position);            // Update positionA
    //p_positionB -> put(positionB);            // Update positionB
    x_position = old_x_position + (position - old_x_position)*sin(theta_max*3.14159/180);
    y_position = old_y_position + (position - *cos(theta_max*3.14159/180);
    //x_position = old_x_position + position*sin(30*3.14159/180);
    //y_position = old_y_position + position*cos(30*3.14159/180);
    p_x_position -> put(x_position);
    p_y_position -> put(y_position);
  }
```

```
  if((movement == 1) && (mode == 0))
  {
    position = count*0.26016;
    p_position -> put(position);            // Update positionA
    //p_positionB -> put(positionB);            // Update positionB
    x_position = old_x_position - position*sin(theta_max*3.14159/180);
    y_position = old_y_position - position*cos(theta_max*3.14159/180);
    //x_position = old_x_position + position*sin(30*3.14159/180);
    //y_position = old_y_position + position*cos(30*3.14159/180);
    p_x_position -> put(x_position);
    p_y_position -> put(y_position);
  }
  if((movement == 2) && (mode == 0))
  {
    position = count*0.26016;
    theta = position*(180/(3.14159*9.878));
    //thetaA = positionA * (180/(3.14159*9.878));
    //thetaB = positionB * (180/(3.14159*9.878));
    //p_positionA -> put(thetaA);
    //p_positionB -> put(thetaB);
    p_position -> put(theta);
    p_theta_max -> put(theta);
    theta = 0;
  }
  if((movement == 3) && (mode == 0))
  {
    position = count*0.26016;
    theta = -position*(180/(3.14159*9.878));        //degrees
    //thetaA = positionA * (180/(3.14159*9.878));
    //thetaB = positionB * (180/(3.14159*9.878));
    //p_positionA -> put(thetaA);
    //p_positionB -> put(thetaB);
    //p_position -> put(theta);
    p_theta_max -> put(theta);
    theta = 0;
  }
}

//*****************************************************************************
/** \brief   This method zeroes out the positions, creating a new zero position.
 *  \details This method resets the position values. This is useful for when the GUI
 *           path tracking plot is reset.
 *  @param   p_serial_port - Pointer to the serial port, used for output.
 *  @return  No return values.
 */

void encoder::zeroposition(void)
{
  uint8_t status = p_zeropositionflag -> get(); // Check the zero position reset flag

  if (status == 1)                              // Verify the flag was set
  {
    p_countA -> put(0);                         // Zero countA
```

```
    p_countB -> put(0);                         // Zero countB
    p_positionA -> put(0);                      // Zero positionA
    p_positionB -> put(0);                      // Zero positionB
    p_position -> put(0);
    p_zeropositionflag -> put(0);               // Reset zero position flag
    p_x_position -> put(0);
    p_y_position -> put(0);
    p_theta_max -> put(0);
  }
}

//*****************************************************************************************
// Declare ISR variables outside for the ISRs for speed
   int16_t powerA = 0;
   int16_t powerB = 0;
   uint8_t modeA = 0;
   uint8_t modeB = 0;

//*****************************************************************************************
 /** \brief   This method services the pulses generated by the encoder for motor B (RIGHT).
 *   \details This ISR depends on whether or not the motors are currently running or not.
 *            This ISR increments the number of interrupts from motor B every time the method
 is called.
 *            Depending on the current motor speed, the reading is recorded as either a count
 increase
 *            or decrease. If the motor speed is zero (occurs also in brake mode) and the encoder
 *            is still generating a pulse, no counts are logged.
 *   @param   INT7_vect - Motor B interrupt.
 *   @return  No return values.
 */

ISR(INT7_vect)
{
    powerA = p_motor_powerA -> ISR_get();
    powerB = p_motor_powerB -> ISR_get();
    modeA = p_modeA -> ISR_get();
    modeB = p_modeB -> ISR_get();
    movement = p_movement -> ISR_get();

    if(movement == 0)               // Forward
    {
      countB = p_countB -> ISR_get();           // Get current value of countB
      countB++;                                 // Increment countB for every interrupt
      p_countB -> ISR_put(countB);              // Put new value of countB in countB
    }
    else if(movement == 1)          // Reverse
    {
      countB = p_countB -> ISR_get();           // Get current value of countB
      countB--;                                 // Increment countB for every interrupt
      p_countB -> ISR_put(countB);
    }
    else if(movement == 2)          // Turning Left
    {
```

```
      countB = p_countB -> ISR_get();           // Get current value of countB
      countB++;                                 // Increment countB for every interrupt
      p_countB -> ISR_put(countB);              // Put new value of countB in countB
    }
    else if(movement == 3)          // Turning Right
    {
      countB = p_countB -> ISR_get();           // Get current value of countB
      countB++;                                 // Increment countB for every interrupt
      p_countB -> ISR_put(countB);              // Put new value of countB in countB
    }
}

//*****************************************************************************
 /** \brief   This method services the pulses generated by the encoder for motor A.
 *   \details This ISR depends on whether or not the motors are currently running or not.
 *            This ISR increments the number of interrupts from motor A every time the method
 is called.
 *            Depending on the current motor speed, the reading is recorded as either a count
 increase
 *            or decrease. If the motor speed is zero (occurs also in brake mode) and the encoder
 *            is still generating a pulse, no counts are logged.
 *   @param   INT1_vect - Motor A intterupt.
 *   @return  No return values.
 */

ISR(INT1_vect)
{
   powerA = p_motor_powerA -> ISR_get();
   powerB = p_motor_powerB -> ISR_get();
   modeA = p_modeA -> ISR_get();
   modeB = p_modeB -> ISR_get();
   movement = p_movement -> ISR_get();

   if(movement == 0)                    // Forward
   {
      countA = p_countA -> ISR_get();           // Get current value of countA
      countA++;                                 // Increment countA for every interrupt
      p_countA -> ISR_put(countA);              // Put new value of countA in countA
   }
   else if(movement == 1)           // Reverse
   {
      countA = p_countA -> ISR_get();           // Get current value of countA
      countA--;                                 // Increment countA for every interrupt
      p_countA -> ISR_put(countA);
   }
   else if(movement == 2)           // Turning Left
   {
      countA = p_countA -> ISR_get();           // Get current value of countA
      countA++;                                 // Increment countA for every interrupt
      p_countA -> ISR_put(countA);              // Put new value of countA in countA
   }
   else if(movement == 3)           // Turning Right
   {
```

```
      countA = p_countA -> ISR_get();           // Get current value of countA
      countA++;                                 // Increment countA for every interrupt
      p_countA -> ISR_put(countA);              // Put new value of countA in countA
   }
}
```

```
//*********************************************************************************
/** \file my_motor_task.h
 *    This file contains the header file for a task which controls the two onboard motor
 drivers. Task
 *    brightness reads the ADC value, and user input from the terminal controls the motor
 *    operation.
 *
 *  Revisions:
 *    \li 03-01-2014 AJB Converted task brightness format to a motor controller
 *    \li 12-01-2014 LBT Updated comments
 *    \li 05-14-2015 LBT Updated comments and formatting
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*********************************************************************************
#ifndef _MY_TASK_H_                         // This define prevents this .h file from being
#define _MY_TASK_H_                         // included multiple times in a .cpp file

#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers
#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues
#include "frt_task.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "frt_queue.h"                      // Header of wrapper for FreeRTOS queues
#include "frt_shared_data.h"                // Header for thread-safe shared data
#include "rs232int.h"                       // ME405/507 library for serial comm.
#include "motor.h"                          // Header for motor class
#include <avr/interrupt.h>                  // Header for interrupts

//*********************************************************************************
/** \brief   This class controls the two onboard motor drivers.
 *  \details The motors are ran using a driver in files motor.h and motor.cpp.
 *           Code sets up and runs the motors.
 */

class my_motor_task : public frt_task
{
    private:
        // Create serial connection for debugging.
```

```
        emstream* ptr_to_serial;

    protected:
        // No protected variables or methods for this class.

    public:
        // This constructor creates a generic task of which many copies can be made
        my_motor_task (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

#endif // _MY_TASK_H_
```

```
//*****************************************************************************
/** \file my_motor_task.cpp
 *  This file contains code for a task which controls the two onboard motor drivers. Task
 *  sensor reads the ADC values by calling adc.cpp, and user input from the MATLAB GUI
 *  controls the motor operation.
 *
 *  Revisions:
 *    \li 03-01-2014 AJB Converted task brightness format to a motor controller
 *    \li 11-29-2014 LBT Updated motor functionality
 *    \li 12-02-2014 LBT Added reverse direction and increase/decrease motor power functionality
 *                       for controlling either one motor or both motors
 *    \li 05-14-2015 LBT Updated comments and formatting
 *    \li 05-31-2015 LBT Removed serial port inputs to methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************
#include "frt_text_queue.h"                     // Header for text queue class
#include "my_motor_task.h"                      // Header for this task
#include "shares.h"                             // Shared inter-task communications
#include "rs232int.h"                           // ME405/507 library for serial comm.
#include <avr/interrupt.h>                      // Header for interrupts

//*****************************************************************************
/** \brief   This constructor creates the motor task.
 *  \details This constructor creates a task which controls the onboard motors through ADC
 readings and
 *           user input.
 *  @param   a_name - A character string which will be the name of this task.
 *  @param   a_priority - The priority at which this task will initially run (default: 0).
 *  @param   a_stack_size - The size of this task's stack in bytes. (default:
 configMINIMAL_STACK_SIZE).
 *  @param   p_ser_dev - Pointer to a serial device (port, radio, SD card, etc.) which can
 *           be used by this task to communicate. (default: NULL).
 *  @return  No return values.
 */

my_motor_task::my_motor_task (const char* a_name, unsigned portBASE_TYPE a_priority, size_t
a_stack_size, emstream* p_ser_dev)
              :frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
```

```
    // Nothing much is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one.
    ptr_to_serial = p_ser_dev;
}

//*****************************************************************************
/** \brief   This method updates the motors for each run through the for (;;) loop.
 *  \details This method is called once by the RTOS scheduler. Each time around the for (;;)
 *           loop, it reads the global control variables and updates the motors accordingly.
 *  @param   No parameters.
 *  @return  No return values.
 */

void my_motor_task::run (void)
{
    portTickType previousTicks = xTaskGetTickCount ();  // Make a variable which will hold times
                                                        // to use for precise task scheduling.
    motor *motor1 = new motor(1);                       // Declare two new motors of class motor.
    motor *motor2 = new motor(2);
    int16_t speedA = 150;                               // Initialize the motor A speed (50%
    duty cycle)
    int16_t speedB = -200;                              // Initialize the motor B speed (50%
    duty cycle)
    p_motor -> put(3);                                  // Initialize the motor number to both
    motors
    p_modeA -> put(1);                                  // Initialize the motor B mode to brake
    p_modeB -> put(1);                                  // Initialize the motor B mode to brake
    p_motor_powerA -> put(0);                           // Initialize the motors to be set at
    brake
    p_motor_powerB -> put(0);                           // Initialize the motors to be set at
    brake
    p_movement -> put(0);                               // Initialize movement information as
    straight
    int increase_power_flag = 0;                        // Initialize increase power flag
    int decrease_power_flag = 0;                        // Initialize decrease power flag
    int reverse_direction_flag = 0;                     // Initialize reverse direction flag
    int turn_left_flag = 0;
    int turn_right_flag = 0;
    int straight_flag = 0;
    int movement = 0;
    int motor_num = 3;
    int motor_modeA = 1;
    int motor_modeB = 1;

    motor1 -> PWM_setup(speedA);                        // Set up the 2 drivers.
    motor2 -> PWM_setup(speedB);

    // This is the task loop for the motor control task. This loop runs until the
    // power is turned off or something equally dramatic occurs.
    for (;;)
    {
        speedA = p_motor_powerA -> get();
        speedB = p_motor_powerB -> get();
```

```
motor_num = p_motor -> get();                          // Get the desired motor number
and mode from
motor_modeA = p_modeA -> get();                        // the user through the
terminal program
motor_modeB = p_modeB -> get();
increase_power_flag = p_increase_power_flag -> get();      // Check current increase
power flag status
decrease_power_flag = p_decrease_power_flag -> get();      // Check current decrease
power flag status
reverse_direction_flag = p_reverse_direction_flag -> get(); // Check current reverse
direction flag status
turn_left_flag = p_turn_left_flag -> get();
turn_right_flag = p_turn_right_flag -> get();
straight_flag = p_straight_flag -> get();
movement = p_movement -> get();

if((motor_num == 1) && (motor_modeA == 1))                 // Brake Motor A (Not used)
{
  motor1 -> brake();
}

if((motor_num == 1) && (motor_modeA == 0))                 // Power Motor A (Not used)
{
  motor1 -> set_power(speedA);
}

if((motor_num == 2) && (motor_modeB ==1))                  // Brake Motor B (Not used)
{
  motor2 -> brake();
}

if((motor_num == 3) && (motor_modeA == 1) && (motor_modeB == 1))// Brake both motors
{
  motor1 -> brake();
  motor2 -> brake();
}

if((motor_num == 3) && (motor_modeA == 0) && (motor_modeB == 0))// Power both motors
{
  motor1 -> set_power(speedA);
  motor2 -> set_power(speedB);
}

if((motor_num == 2) && (motor_modeB ==1))                  // Brake Motor B (Not used)
{
  motor2 -> brake();
}

if((motor_num == 2) && (motor_modeB == 0))                 // Power Motor B (Not used)
{
  motor2 -> set_power(speedB);
}
```

```
    if((increase_power_flag == 1) && (motor_num == 1))          // Increase Motor A power
    (Not used)
    {
      motor1 -> increase_power(speedA);
    }

    if((decrease_power_flag == 1) && (motor_num == 1))          // Decrease Motor A power
    (Not used)
    {
      motor1 -> decrease_power(speedA);
    }

    if((increase_power_flag == 1) && (motor_num == 2))          // Increase Motor B power
    (Not used)
    {
      motor2 -> increase_power(speedB);
    }

    if((decrease_power_flag == 1) && (motor_num == 2))          // Decrease Motor B power
    (Not used)
    {
      motor2 -> decrease_power(speedB);
    }

    if((reverse_direction_flag == 1) && (motor_num == 1))       // Reverse Motor A
    direction (Not used)
    {
      motor1 -> reverse_direction(speedA);
    }
    if((reverse_direction_flag == 1) && (motor_num == 2))       // Reverse Motor B
    direction (Not used)
    {
      motor2 -> reverse_direction(speedB);
    }

    if((reverse_direction_flag == 1) && (motor_num == 3))       // Reverse both motors
    direction
    {
      motor1 -> reverse_direction(speedA);
      p_reverse_direction_flag -> put(1);                       // Set flag for second call
      to run
      motor2 -> reverse_direction(speedB);
      p_countA -> put(0);
      p_countB -> put(0);
      p_movement -> put(1);
      p_position -> put(0);
    }

    if((increase_power_flag == 1) && (motor_num == 3))          // Increase both motors power
    {
      motor1 -> increase_power(speedA);
      p_increase_power_flag -> put(1);                          // Set flag for second call
      to run
```

```
    motor2 -> increase_power(speedB);
  }

  if((decrease_power_flag == 1) && (motor_num == 3))          // Decrease both motors power
  {
    motor1 -> decrease_power(speedA);
    p_decrease_power_flag -> put(1);                          // Set flag for second call
    to run
    motor2 -> decrease_power(speedB);
  }

  if(turn_left_flag == 1)
  {
    p_reverse_direction_flag -> put(1);


    if(speedA < 0)
    {
      motor1 -> reverse_direction(speedA);
    }

   p_reverse_direction_flag -> put(1);

   if(speedB > 0)
    {
      motor2 -> reverse_direction(speedB);
    }
    speedA = p_motor_powerA -> get();
    p_reverse_direction_flag -> put(1);
    motor1 -> reverse_direction(speedA);
    p_countA -> put(0);
    p_countB -> put(0);
    p_turn_left_flag -> put(0);
    p_movement -> put(2);
    p_position -> put(0);
  }

  if(turn_right_flag == 1)
  {
    p_reverse_direction_flag -> put(1);


    if(speedA < 0)
    {
      motor1 -> reverse_direction(speedA);
    }

   p_reverse_direction_flag -> put(1);

   if(speedB > 0)
    {
      motor2 -> reverse_direction(speedB);
    }
```

```
          speedB = p_motor_powerB -> get();
          p_reverse_direction_flag -> put(1);
          motor2 -> reverse_direction(speedB);
          p_countA -> put(0);
          p_countB -> put(0);
          p_turn_right_flag -> put(0);
          p_movement -> put(3);
          p_position -> put(0);
        }

        if(straight_flag == 1)
        {
          p_reverse_direction_flag -> put(1);

          if(speedA < 0)
          {
            motor1 -> reverse_direction(speedA);
          }

         p_reverse_direction_flag -> put(1);
          if(speedB > 0)
          {
            motor2 -> reverse_direction(speedB);
          }

          p_countA -> put(0);
          p_countB -> put(0);
          p_reverse_direction_flag -> put(0);
          p_straight_flag -> put(0);
          p_movement -> put(0);
          p_position -> put(0);
        }

        // Increment the run counter. This counter belongs to the parent class and can
        // be printed out for debugging purposes.
        runs++;

        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times.
        delay_from_to (previousTicks, configMS_TO_TICKS (100));
    }
}
```

```
//*****************************************************************************
/** \file motor.h
 *    This file contains definitions for the motor drivers.
 *
 *  Revisions:
 *    \li 01-28-2014 AJB Original file designed
 *    \li 12-02-2014 LBT Added reverse direction and increase/decrease motor power methods
 *    \li 05-13-2015 LBT Updated comments and formatting
 *    \li 05-29-2015 LBT Removed freewheel function
 *    \li 05-31-2015 LBT Removed serial port inputs to methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************
#ifndef _AVR_MOTOR_H_                       // This define prevents this .H file from being
#define _AVR_MOTOR_H_                       // included multiple times in a .CPP file

#include <stdlib.h>                         // Include standard library header files
#include <avr/io.h>                         // Include Port I/O for SFR's
#include "emstream.h"                       // Header for serial ports and devices
#include "FreeRTOS.h"                       // Header for the FreeRTOS RTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // Header for FreeRTOS queues
#include "semphr.h"                         // Header for FreeRTOS semaphores

//*****************************************************************************
/** \brief   This class should run the motor drivers for two motors.
 *  \details The Wall Climbing Robot board has two motor drivers. This class enables them.
 */

class motor
{
    private:
        // No private variables or methods for this class.

    protected:
        // The ADC class uses this pointer to the serial port to communicate.
        emstream* ptr_to_serial;

        // Create a variable for the selected motor driver.
        int driver_num;
```

```
    public:
        // Constructor for motor class.
        motor (int);

        // Sets up the PWM output connected to the motor drivers.
        void PWM_setup (int16_t);

        // Updates the motor speed.
        void set_power(int16_t);

        // Updates the motor speed.
        void reverse_direction(int16_t);

        // Increases the motor speed.
        void increase_power(int16_t);

        // Decreases the motor speed.
        void decrease_power(int16_t);

        // Stops rotation.
        void brake(void);

}; // end of class motor

#endif // _AVR_MOTOR_H_
```

```
//*********************************************************************************
/** \file motor.cpp
 *    This file contains code for two simple motor drivers using PWM.
 *
 *  Revisions:
 *    \li 01-28-2014 AJB Created basic motor driver
 *    \li 11-29-2012 LBT Updated motor ports and pins
 *    \li 12-02-2014 LBT Added reverse direction and increase/decrease motor power methods
 *    \li 12-05-2014 LBT Updated comments
 *    \li 05-14-2015 LBT Updated comments and formatting
 *    \li 05-29-2015 LBT Removed freewheel function
 *    \li 05-31-2015 LBT Removed serial port inputs to methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*********************************************************************************
#include <stdlib.h>                     // Include standard library header files
#include <avr/io.h>                     // Include Port I/O for SFR's
#include "rs232int.h"                   // Include header for serial port class
#include "motor.h"                      // Include header for motor class
#include "my_motor_task.h"              // Include header for my_motor_task class
#include "frt_text_queue.h"             // Header for text queue class
#include "shares.h"                     // Shared inter-task communications

//*********************************************************************************
/** \brief   This constructor sets up and initializes the motor drivers.
 *  \details The board has 2 motor drivers. The user can select which one
 *           is to be used. This constructor declares the one important
 *           driver parameter, which is which driver on the board is being used.
 *           Finally, a message can be printed to note the end of setup.
 *  @param   desired_driver - Target driver selected in my_motor_task.
 *  @param   p_serial_port - A pointer to the serial port which writes debugging info.
 *  @return  No return values
 */

motor::motor(int desired_driver)
{
    driver_num = desired_driver;                // Create name for passed variable
}

//*********************************************************************************
```

```
/** \brief   This method sets up one of the motor drivers.
 *  \details This method sets up the PWM mode of the microcontroller, the clock speed,
 *           selects the output pins, and the PWM ports. It also enables the drivers and the
 initial
 *           directions of the motors.
 *  @param   speed - Initial motor power set in my_motor_task.
 *  @return  No return values.
 */

void motor::PWM_setup (int16_t speed)           // Set up PWM for motor drivers
{
    TCCR1A  |=   (1<<WGM10);                    // Setting fast PWM for motor driver 1 (mode 5)
    TCCR1B  |=   (1<<WGM12)|                    // Mode 5 (page 148)
                 (1<<CS11)|
                 (1<<CS10);                     // Set prescaler to 64 (page 161)
    TCCR2A  |=   (1<<WGM21)|                    // Setting fast PWM for motor driver 1 (mode 3)
                 (1<<WGM20);                    // Mode 3 (page 189)
    TCCR2B  |=   (1<<CS22);                     // Set prescaler to 64 (page 191)
    if(driver_num == 1)                         // Motor driver 1 (Motor A)
    {
      TCCR1A    |=  (1<<COM1A1);                // COMnBx enables output compare mode on
      PortB5, OC1A (non-inverting mode)
      TCCR1A    &= ~(1<<COM1A0);
      DDRC      |=  (1<<PC5)|                   // Enable the control output pins
                    (1<<PC6)|
                    (1<<PC7);
      DDRB      |=  (1<<PB5);                   // Enable PWM output
      OCR1A      =  speed;                      // Timer counter compare default value
      PORTC     |=  (1<<PC6)|                   // Set default rotation forward
                    (1<<PC5);                   // Enable the driver
      p_motor_powerA -> put(speed);             // Update current speed value of motor A
    }
    if(driver_num == 2)                         // Motor driver 2 (Motor B)
    {
      TCCR2A    |=  (1<<COM2A1);                // COMnBx enables output compare mode on
      PortB4, OC2A (non-inverting mode)
      TCCR2A    &= ~(1<<COM2A0);
      DDRC      |=  (1<<PC2)|                   // Enable the control output pins
                    (1<<PC3)|
                    (1<<PC4);
      DDRB      |=  (1<<PB4);                   // Enable PWM output
      OCR2A      =  speed;                      // Timer counter compare default value
      PORTC     |=  (1<<PC4)|                   // Set default rotation forward
                    (1<<PC2);                   // Enable the driver
      p_motor_powerB -> put(speed);             // Update current speed value of motor B
    }
}

//*****************************************************************************
/** \brief   This method sets the motor powers through PWM.
 *  \details This method takes in a 16 bit number, and controls the motor speeds with it.
 *           Negative and positive numbers result in motions in opposite directions. Motion
 *           is controlled by updating the output compare resister.
```

```
 *  @param   speed - Motor speed where smaller numbers result in slower motion.
 *  @return  No return values.
 */

void motor::set_power(int16_t speed)
{
    if(driver_num == 1)                 // Motor driver 1
    {
        OCR1A = abs(speed);             // Update motor speed
        if(speed < 0)                   // Set motor to rotate CCW Reverse  CHECK DIRECTIONS
        {
            PORTC |=   (1 << PC7);
            PORTC &=  ~(1 << PC6);
        }
        else                            // Set motor to rotate CW Forward   CHECK DIRECTIONS
        {
            PORTC |=   (1 << PC6);
            PORTC &=  ~(1 << PC7);
        }
    }
    else                                // Motor driver 2
    {
        OCR2A = abs(speed);             // Update motor speed
        if(speed < 0)                   // Set motor to rotate CW Forward CHECK DIRECTIONS
        {
            PORTC |=   (1 << PC4);
            PORTC &=  ~(1 << PC3);
        }
        else                            // Set motor to rotate CCW Reverse CHECK DIRECTIONS
        {
            PORTC |=   (1 << PC3);
            PORTC &=  ~(1 << PC4);
        }
    }
}

//*****************************************************************************
/** \brief   This method reverses the direction of the motors through PWM.
 *  \details This method takes in a 16 bit number, and controls the motor speeds with it.
 *           Negative and positive numbers result in motions in opposite directions. Motion
 *           is controlled by updating the output compare resister.
 *  @param   speed -  Motor speed where smaller numbers result in slower motion.
 *  @return  No return values.
 */

void motor::reverse_direction(int16_t speed)
{
  uint8_t status = p_reverse_direction_flag -> get();// Check the reverse direction flag
  if (status == 1)                      // If the reverse direction flag is set, continue
  {
    if(driver_num == 1)                 // Motor driver 1
    {
        if(speed > 0)                   // Set motor to rotate CW from CCW  CHECK DIRECTIONS
```

```
        {
            PORTC |=   (1 << PC7);
            PORTC &=  ~(1 << PC6);
        }
        else                            // Set motor to rotate CCW from CW  CHECK DIRECTIONS
        {
            PORTC |=   (1 << PC6);
            PORTC &=  ~(1 << PC7);
        }
        speed = -1*speed;               // Invert speed value
        OCR1A = abs(speed);             // Update motor speed
        p_motor_powerA -> put(speed);
    }
    else                                // Motor driver 2
    {
        if(speed > 0)                   // Set motor to rotate CW from CCW
        {
            PORTC |=   (1 << PC4);
            PORTC &=  ~(1 << PC3);
        }
        else                            // Set motor to rotate CCW from CW
        {
            PORTC |=   (1 << PC3);
            PORTC &=  ~(1 << PC4);
        }
        speed = -1*speed;               // Invert speed value
        OCR2A = abs(speed);             // Update motor speed
        p_motor_powerB -> put(speed);
    }
    p_reverse_direction_flag -> put(0); // Reset reverse direction flag
  }
}

//*****************************************************************************
/** \brief   This method increases the motor powers through PWM.
 *  \details This method takes in a 16 bit number, and controls the motor speed with it.
 *           Negative and positive numbers result in motions in opposite directions. Motion
 *           is controlled by updating the output compare resister.
 *  @param   speed - Motor speed where smaller numbers result in slower motion.
 *  @param   p_serial_port - Pointer to the serial port, used for output.
 *  @return  No return values.
 */

void motor::increase_power(int16_t speed)
{
  uint8_t status = p_increase_power_flag -> get();  // Check the increase power flag
  if (status == 1)                                  // If the increase power flag is set,
  continue
  {
    int32_t new_speed;                              // Create new_speed variable
    if(driver_num == 1)                             // Motor driver 1
    {
        if(speed >= 0)                              // If motor speed is positive or zero
```

```
        {
            new_speed = speed + 10;                 // Increase speed by 10
            if(abs(new_speed) >= 245)               // If motor speed has reached or exceeded
            the maximum value
            {
                new_speed = 245;                    // Cap the speed at the maximum value and
                print message
            }
        }
        else                                        // If motor speed is negative
        {
            new_speed = speed - 10;                 // Increase speed by 10
            if(abs(new_speed) >= 245)               // If motor speed has reached or exceeded
            the maximum value
            {
                new_speed = -245;                   // Cap the speed at the maximum value and
                print message
            }
        }
        OCR1A = abs(new_speed);                     // Update motor speed
        p_motor_powerA -> put(new_speed);
        if(new_speed < 0)                           // Set motor to rotate CCW reverse
        {
            PORTC |=   (1 << PC7);
            PORTC &=  ~(1 << PC6);
        }
        else                                        // Set motor to rotate CW forward
        {
            PORTC |=   (1 << PC6);
            PORTC &=  ~(1 << PC7);
        }
    }
    else                                            // Motor driver 2
    {
        if(speed >= 0)                              // If motor speed is positive or zero
        {
            new_speed = speed + 10;                 // Increase speed by 10
            if(abs(new_speed) >= 245)               // If motor speed has reached or exceeded
            the maximum value
            {
                new_speed = 245;                    // Cap the speed at the maximum value and
                print message
            }
        }
        else                                        // If motor speed is negative
        {
            new_speed = speed - 10;                 // Increase speed by 10
            if(abs(new_speed) >= 245)               // If the motor speed has reached or
            exceeded the maximum value
            {
                new_speed = -245;                   // Cap the speed at the maximum value and
                print message
            }
```

```
        }
        OCR2A = abs(new_speed);                         // Update motor speed
        p_motor_powerB -> put(new_speed);
        if(new_speed < 0)                               // Set motor to rotate CW forward
        {
            PORTC |=   (1 << PC4);
            PORTC &=  ~(1 << PC3);
        }
        else                                            // Set motor to rotate CCW reverse
        {
            PORTC |=   (1 << PC3);
            PORTC &=  ~(1 << PC4);
        }
    }
    p_increase_power_flag -> put(0);                    // Reset increase power flag
  }
}

//*****************************************************************************
/** \brief   This method decreases the motor powers through PWM.
 *  \details This method takes in a 16 bit number, and controls the motor speed with it.
 *           Negative and positive numbers result in motions in opposite directions. Motion
 *           is controlled by updating the output compare resister.
 *  @param   speed - Motor speed where smaller numbers result in slower motion.
 *  @param   p_serial_port - Pointer to the serial port, used for output.
 *  @return  No return values.
 */

void motor::decrease_power(int16_t speed)
{
  uint8_t status = p_decrease_power_flag -> get();  // Check the decrease power flag
  if (status == 1)                                  // If the decrease power flag is set,
  continue
  {
    int16_t new_speed;                              // Create new_speed variable
    if(driver_num == 1)                             // Motor driver 1
    {
        if(speed >= 0)                              // If motor speed is positive or zero
        {
            new_speed = speed - 10;                 // Decrease speed by 10
            if(abs(new_speed) >= abs(speed))        // If motor speed has reached within -5 or 5
            {
                new_speed = 0;                      // Set the speed to the minimum value and
                print message
            }
        }
        else                                        // If motor speed is negative
        {
            new_speed = speed + 10;                 // Decrease speed by 10
            if(abs(new_speed) >= abs(speed))        // If motor speed has reached within -5 and 5
            {
                new_speed = 0;                      // Set the speed to the minimum value and
                print message
```

```
            }
        }
        OCR1A = abs(new_speed);                     // Update motor speed
        p_motor_powerA -> put(new_speed);
        if(speed < 0)                               // Set motor to rotate CCW reverse
        {
            PORTC |=   (1 << PC7);
            PORTC &=  ~(1 << PC6);
        }
        else                                        // Set motor to rotate CW forward
        {
            PORTC |=   (1 << PC6);
            PORTC &=  ~(1 << PC7);
        }
    }
    else                                            // Motor driver 2
    {
        if(speed >= 0)                              // If motor speed is positive or zero
        {
            new_speed = speed - 10;                 // Decrease speed by 10
            if (abs(new_speed) > abs(speed))        // If motor speed has reached within -5 and 5
            {
                new_speed = 0;                      // Set the speed to the minimum value and
                print message
            }
        }
        else                                        // If motor speed is negative
        {
            new_speed = speed + 10;                 // Decrease speed by 10
            if(abs(new_speed) >= abs(speed))        // If motor speed has reached within -5 and 5
            {
                new_speed = 0;                      // Set the speed to the minimum value and
                print message
            }
        }
        OCR2A = abs(new_speed);                     // Update motor speed
        p_motor_powerB -> put(new_speed);
        if(speed < 0)                               // Set motor to rotate CW forward
        {
            PORTC |=   (1 << PC4);
            PORTC &=  ~(1 << PC3);
        }
        else                                        // Set motor to rotate CCW reverse
        {
            PORTC |=   (1 << PC3);
            PORTC &=  ~(1 << PC4);
        }
    }
    p_decrease_power_flag -> put(0);                // Reset decrease power flag
  }
}

//*****************************************************************************
```

```
/** \brief   This method brakes the motor.
 *  \details This method stops the motor rotations by shorting the motor pins.
 *  @param   No parameters.
 *  @return  No return values.
 */

void motor::brake(void)
{
    if(driver_num == 1)             // Motor driver 1
    {
      PORTC |=  (1 << PC6);         // Brake motor to VCC
      PORTC |=  (1 << PC7);
    }
    if(driver_num == 2)             // Motor driver 2
    {
      PORTC |=  (1 << PC3);         // Brake motor to VCC
      PORTC |=  (1 << PC4);
    }
}
```

```
//*****************************************************************************
/** \file task_user.h
 *    This file contains header inforamtion for a user interface task for a Wall-Climbing
 *    Robot project.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task
 *    \li 10-25-2012 JRR Changed to a more fully C++ version with class task_user
 *    \li 11-04-2012 JRR Modified from the data acquisition example to the test suite
 *    \li 12-01-2014 LBT Added commands for controlling both or one motor and increasing/
 *                       decreasing motor speeds and displaying current motor speeds
 *    \li 05-31-2015 LBT Updated formatting and comments, removed help message and
 *                       status methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************
#ifndef _TASK_USER_H_                       // This define prevents this .h file from being
#define _TASK_USER_H_                       // included multiple times in a .cpp file

#include <stdlib.h>                         // Prototype declarations for I/O functions
#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues
#include "rs232int.h"                       // ME405/507 library for serial comm.
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "frt_task.h"                       // Header for ME405/507 base task class
#include "frt_queue.h"                      // Header of wrapper for FreeRTOS queues
#include "frt_text_queue.h"                 // Header for a "<<" queue class
#include "frt_shared_data.h"                // Header for thread-safe shared data
#include "shares.h"                         // Global ('extern') queue declarations

/// This macro defines a string that identifies the name and version of this program.
#define PROGRAM_VERSION                     PMS ("Wall-Climbing Robot Program V1.1 ")

//---------------------------------------------------------------------------
/** \brief   This class controls the graphical user interface.
 *  \details This class interacts with the graphical user interface and controls the inputs
 *           and outputs to that interface.
 */
```

```
class task_user : public frt_task
{
private:
    // No private variables or methods for this class

protected:
    // No protected variables or methods for this class

public:
    // This constructor creates a user interface task object.
    task_user (const char*, unsigned portBASE_TYPE, size_t, emstream* p_ser_dev);

    // This method is called by the RTOS once to run the task loop for ever and ever.
    void run (void);
};

#endif // _TASK_USER_H_
```

```
//****************************************************************************
/** \file task_user.cpp
 *    This file contains source code for a user interface task for a Wall-Climbing Robot
 *    project.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task
 *    \li 10-25-2012 JRR Changed to a more fully C++ version with class task_user
 *    \li 11-04-2012 JRR Modified from the data acquisition example to the test suite
 *    \li 01-03-2014 AJB Modified to include user input for the motors
 *    \li 02-11-2014 LBT Modified to include new user input for the encoder tasks
 *    \li 11-29-2014 LBT Updated to include functionality for two motors and encoders
 *    \li 12-02-2014 LBT Added user inputs for reversing motor direction, increasing/decreasing
 *                       motor power, and controlling either one motor or both motors
 *    \li 05-31-2015 LBT Updated formatting and comments, removed help message and
 *                       status methods
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************
#include <avr/io.h>                             // Port I/O for SFR's
#include <avr/wdt.h>                            // Watchdog timer header
#include "shared_data_sender.h"                 // Enable shared data
#include "shared_data_receiver.h"               // Enable received data
#include "task_user.h"                          // Header for this file
#include "shares.h"                             // Shared inter-task communications
#include "math.h"

//****************************************************************************
/** \brief   This constant determines the task delay.
 *  \details This constant sets how many RTOS ticks the task delays if the user's not talking.
 *           The duration is calculated to be about 5 ms.
 *  @param   No parameters
 *  @return  No return values
 */
const portTickType ticks_to_delay = ((configTICK_RATE_HZ / 1000) * 5);

//****************************************************************************
/** \brief   This constructor creates a new data acquisition task.
 *  \details Its main job is to call the parent class' constructor, which does most of the work.
```

```
 *  @param   a_name A character string which will be the name of this task.
 *  @param   a_priority The priority at which this task will initially run (default: 1).
 *  @param   a_stack_size The size of this task's stack in bytes. (default:
 configMINIMAL_STACK_SIZE).
 *  @param   p_ser_dev Pointer to a Bluetooth serial device which can be used by this task to
 *           communicate. (default: bluetooth_ser_port).
 *  @return  No return values.
 */

task_user::task_user (const char* a_name, unsigned portBASE_TYPE a_priority, size_t a_stack_size
, emstream* p_ser_dev)
          :frt_task (a_name, a_priority, a_stack_size, p_ser_dev)
{
    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}

//---------------------------------------------------------------------------
/** \brief   This task runs the graphical user interface.
 *  \details This task interacts with the user to control the inputs and outputs to the GUI.
 *  @param   No parameters.
 *  @return  No return values.
 */

void task_user::run (void)
{
    int motor;                              // Current motor being controlled
    double sensor1;                         // Create variables for sensor values
    double sensor2;
    double sensor3;
    double sensor4;
    double sensor5;
    double sensor6;
    double sensor7;
    double sensor8;
    double sensor9;                         // Proximity sensors
    double sensor10;
    double x_position;                      // Current positions
    double y_position;
    double position;
    double proximity;
    double speed;                           // Current speed
    char char_in;                           // Character read from serial device

    // This is an infinite loop; it runs until the power is turned off. There is one
    // such loop inside the code for each task.
    for (;;)
    {
        // Run the finite state machine. The variable 'state' is kept by the parent class.
        switch (state)
        {
            // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            // In state 0, we're in command mode, so when the GUI sends characters, the
```

```
        // characters are interpreted as commands to do something.
        case (0):
            if (p_serial->check_for_char ())    // If the user typed a
            {                                   // character, read
                char_in = p_serial->getchar (); // the character

                // In this switch statement, we respond to different characters as
                // commands typed in by the user.
                switch (char_in)
                {
                    // The '1' command sets motor A as current motor. (Not used, but could
                    be used to change turning scheme)
                    case ('1'):
                        p_motor -> put(1);
                        break;

                    // The '2' command sets motor B as current motor. (Not used, but could
                    be used to change turning scheme)
                    case ('2'):
                        p_motor -> put(2);
                        break;

                    // The '3' command sets both motors as current motor. (Default)
                    case ('3'):
                        p_motor -> put(3);
                        break;

                    // The 'p' command sets current motor to power mode. ("Power motors" on
                    GUI)
                    case ('p'):
                        motor = p_motor -> get();
                        if(motor == 1)
                        {
                          p_modeA -> put(0);
                        }
                        if(motor == 2)
                        {
                          p_modeB -> put(0);
                        }
                        if(motor == 3)              // One that is used
                        {
                          p_modeA -> put(0);
                          p_modeB -> put(0);
                        }
                        break;

                    // The 'b' command sets current motor to brake mode. ("Brake Motors" on
                    GUI)
                    case ('b'):
                        motor = p_motor -> get();
                        if(motor == 1)
                        {
                          p_modeA -> put(1);
```

```
        }
        if(motor == 2)
        {
          p_modeB -> put(1);
        }
        if(motor == 3)              // One that is used
        {
          p_modeA -> put(1);
          p_modeB -> put(1);
        }
        break;

    // The 'z' command resets position counts. ("Reset Path Plot" on GUI)
    case ('z'):
        p_zeropositionflag -> put(1);
        break;

    // The 'r' command reverses the current motor direction. ("Reverse" on
    GUI)
    case ('r'):
        p_reverse_direction_flag -> put(1);
        break;

    // The 'i' command increases the power for the current motor.
    ("Increase Speed" on GUI)
    case ('i'):
        p_increase_power_flag -> put(1);
        break;

    // The 'd' command decreases the power for the current motor.
    ("Decrease Speed" on GUI)
    case ('d'):
        p_decrease_power_flag -> put(1);
        break;

    // The 'l' command turns the robot to the left. ("Turn Left" on GUI)
    case ('l'):
        p_turn_left_flag -> put(1);
        break;

    // The 'g' command turns the robot to the right. ("Turn Right" on GUI)
    case ('g'):
        p_turn_right_flag -> put(1);
        break;

    // The 's' command turns the robot straight. ("Straight" on GUI)
    case ('s'):
        p_straight_flag -> put(1);
        break;

    case ('a'):
        sensor1 = p_sensor1 -> get();       // Get current sensor
        values
```

```
                        sensor2 = p_sensor2 -> get();
                        sensor3 = p_sensor3 -> get();
                        sensor4 = p_sensor4 -> get();
                        sensor5 = p_sensor5 -> get();
                        sensor6 = p_sensor6 -> get();
                        sensor7 = p_sensor7 -> get();
                        sensor8 = p_sensor8 -> get();
                        proximity = p_proximity -> get();       // Proximity sensors
                        x_position = p_x_position -> get(); // Current positions
                        y_position = p_x_position -> get();
                        speed = p_speed -> get();           // Current speed


                        *p_serial << sensor1 << PMS(", ") << sensor2 << PMS(", ") << sensor3
                         << PMS(", ")
                                  << sensor4 << PMS(", ") << sensor5 << PMS(", ") << sensor6
                                   << PMS(", ")
                                  << sensor7 << PMS(", ") << sensor8 << PMS(", ") << speed
                                  << PMS(", ")
                                  << x_position << PMS(", ") << y_position <<  PMS(", ") <<
                                  proximity << endl;
                        break;

                    /* case ('c'):
                        x_position = p_x_position -> get();
                        y_position = p_y_position -> get();
                        *p_serial << x_position << PMS(", ") << y_position << endl;
                        break; */

                    // If the character isn't recognized, ignore it.
                    default:
                        break;
                }; // End switch for characters
            } // End if a character was received

            break; // End of state 0

        // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        // We should never get to the default state. If we do, complain and restart
        default:
            wdt_enable (WDTO_120MS);
            break;
    } // End switch state

    runs++;                             // Increment counter for debugging

    // No matter the state, wait for approximately a millisecond before we
    // run the loop again. This gives lower priority tasks a chance to run
    vTaskDelay (configMS_TO_TICKS (1));
  }
}

//*****************************************************************************
```

```
/** \brief   This method prints a simple help message.
 *  \details This method displays the functions of each key in the terminal.
 *  @param   No parameters
 *  @return  No return values
 */

//void task_user::print_help_message (void)     //instructions for user input
//{
    //*p_serial << PMS ("    1:   Control Motor A") << endl;
    //*p_serial << PMS ("    2:   Control Motor B") << endl;
    //*p_serial << PMS ("    3:   Control both motors") << endl;
    //*p_serial << PMS ("    b:   Brake current motor(s)") << endl;
    //*p_serial << PMS ("    f:   Freewheel current motor(s)") << endl;
    //*p_serial << PMS ("    p:   Power current motor(s)") << endl;
    //*p_serial << PMS ("    z:   Zero motor position and error count") << endl;
    //*p_serial << PMS ("    r:   Reverse current motor direction ") << endl;
    //*p_serial << PMS ("    i:   Increase current motor power") << endl;
    //*p_serial << PMS ("    d:   Decrease current motor power") << endl;
    //*p_serial << PMS ("    l:   Turn robot left") << endl;
    //*p_serial << PMS ("    g:   Turn robot right") << endl;
    //*p_serial << PMS ("    s:   Robot goes straight") << endl;
    //*p_serial << PMS ("    a:   Output current sensor readings") << endl;
//}

//*****************************************************************************
/** \brief   This method displays the current system status information.
 *  \details This method displays information about the status of the system, including the
 following:
 *           \li The name and version of the program.
 *           \li The name, status, priority, and free stack space of each task.
 *           \li Processor cycles used by each task.
 *           \li Amount of heap space free and setting of RTOS tick timer.
 *  @param   No parameters.
 *  @return  No return values.
 *
 */

//void task_user::show_status (void)
//{
    //time_stamp the_time;          // Holds current time for printing

    // Show program vesion, time, and free heap space
    //*p_serial << endl << PROGRAM_VERSION << PMS (__DATE__) << endl
        //    << PMS ("Time: ") << the_time.set_to_now ()
            //  << PMS (", Heap free: ") << heap_left() << PMS ("/")
             // << configTOTAL_HEAP_SIZE;

    // Show how the timer/counter is set up to cause RTOS timer ticks
    //#if (defined OCR5A)
        //*p_serial << PMS (", OCR5A=") << OCR5A << endl << endl;
    //#elif (defined OCR3A)
        //*p_serial << PMS (", OCR3A=") << OCR3A << endl << endl;
    //#else
```

```
        //*p_serial << PMS (", OCR1A=") << OCR1A << endl << endl;
    //#endif

    // Have the tasks print their status
    //print_task_list (p_serial);
//}
```

C. OVERALL TASK DIAGRAM FOR C++ CODE

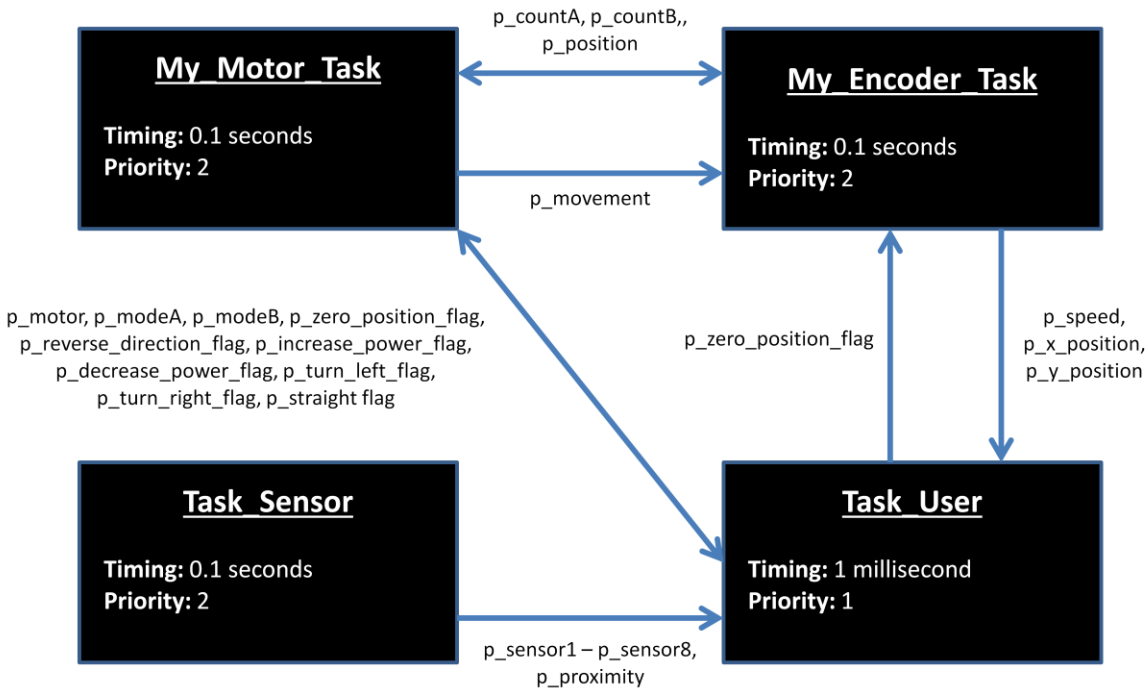

Figure 24: Overall task diagram for final C++ code

Table 10: Data types for shared variables in overall task diagram

Shared Variable Name	Data Type	Shared Variable Name	Data Type
p_countA	int32_t	p_motor	uint8_t
p_countB	int32_t	p_modeA	unit8_t
p_position	double	p_modeB	unit8_t
p_movement	uint8_t	p_zero_position_flag	unit8_t
p_speed	double	p_reverse_direction_flag	unit8_t
p_sensor1	double	p_increase_power_flag	unit8_t
p_sensor2	double	p_decrease_power_flag	unit8_t
p_sensor3	double	p_turn_left_flag	unit8_t
p_sensor4	double	p_turn_right_flag	unit8_t
p_sensor5	double	p_straight_flag	unit8_t
p_sensor6	double	p_proximity	double
p_sensor7	double	p_x_position	double
p_sensor8	double	p_y_position	double

D. CALCULATIONS IN C++ CODE EXECUTION

ADC Conversion

$$Voltage\ Output = \frac{V_S * ADC\ Output}{2^{10} - 1}\ [Volts], where\ V_S\ is\ the\ ADC\ supply\ voltage$$

Encoder Resolution

$$Resolution = \frac{\pi D}{\#\ Gear\ Teeth}\ \left[\frac{in}{tooth}\right], where\ D\ is\ the\ drive\ wheel\ diameter$$

Linear Position

$$l = Resolution * Encoder\ Count\ [in]$$

Linear Speed

$$Linear\ Speed = \frac{\Delta Linear\ Position}{0.1\ seconds}\ \left[\frac{in}{s}\right], where\ 0.1\ seconds\ is\ the\ code\ task\ run\ time$$

Angular Position

$$\theta = Resolution * Encoder\ Count * \frac{180°}{\pi R}\ [degrees], where\ R\ is\ the\ skid\ turn\ radius$$

Angular Speed

$$\dot{\theta} = \frac{\Delta\theta}{0.1\ seconds}\ \left[\frac{degrees}{s}\right], where\ 0.1\ seconds\ is\ the\ code\ task\ run\ time$$

Current Position

$$x\ position = old\ x\ position + lsin(\theta_{\max})\ \ [in]$$

$$y\ position = old\ y\ position + lcos(\theta_{\max})\ \ [in]$$

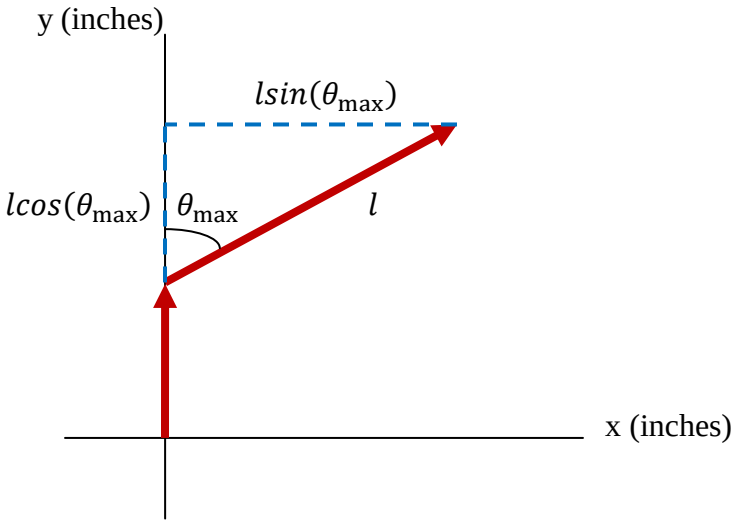

Figure 25: Robot sample motion diagram used to determine x and y coordinate positions

E. MATLAB GRAPHICAL USER INTERFACE CODE

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

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

% Last Modified by GUIDE v2.5 04-Jun-2015 21:51:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Wall_Climbing_Robot_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @Wall_Climbing_Robot_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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

% --- Executes just before Wall_Climbing_Robot_GUI is made visible.
function Wall_Climbing_Robot_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
```

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

xlabel(handles.axes3,'Time (seconds)');
ylabel(handles.axes3,'Pressure (kPa)');
xlabel(handles.axes4, 'X Position (in)');
ylabel(handles.axes4, 'Y Position (in)');
grid(handles.axes3,'on');
grid(handles.axes4,'on');
set(handles.pushbutton25, 'Enable', 'off');
set(handles.pushbutton23, 'Enable', 'off');
set(handles.pushbutton26, 'Enable', 'off');
set(handles.pushbutton21, 'Enable', 'off');
set(handles.pushbutton12, 'Enable', 'off');
set(handles.pushbutton13, 'Enable', 'off');
set(handles.pushbutton14, 'Enable', 'off');
set(handles.pushbutton4, 'Enable', 'off');
set(handles.pushbutton9, 'Enable', 'off');
set(handles.pushbutton3, 'Enable', 'off');
set(handles.pushbutton1, 'Enable', 'off');
set(handles.pushbutton6, 'Enable', 'off');
set(handles.pushbutton8, 'Enable', 'off');

%instrreset;

%pause(0.1);

global t;
global var;
global pressure;
global i;
global obj1;
global x_position;
global y_position;
global voltage;
global proximity;

i = 1;
pressure = zeros(10000,8);
var.x = 0;
t = zeros(10000,1);
x_position = zeros(10000,1);
y_position = zeros(10000,1);
voltage = zeros(10000,1);
proximity = zeros(10000,1);

obj1 = instrfind('Type', 'bluetooth', 'Name', 'Bluetooth-Wall   RobotSmirf:1', 'Tag',↙
'');
```

```
% Choose default command line output for Wall_Climbing_Robot_GUI
handles.output = hObject;

var.tmr = timer('TimerFcn',{@pushbutton26_Callback,handles},'Period',0.5,...
                'ExecutionMode','FixedRate');

guidata(hObject, handles);

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

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

% Power On --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'p');
set(handles.pushbutton12, 'Enable', 'on');
set(handles.pushbutton13, 'Enable', 'on');
set(handles.pushbutton14, 'Enable', 'on');
set(handles.pushbutton4, 'Enable', 'on');
set(handles.pushbutton9, 'Enable', 'on');
set(handles.pushbutton3, 'Enable', 'on');
set(handles.pushbutton1, 'Enable', 'off');
set(handles.pushbutton6, 'Enable', 'on');
set(handles.pushbutton8, 'Enable', 'on');

% Power Off --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'b');
set(handles.pushbutton12, 'Enable', 'off');
set(handles.pushbutton13, 'Enable', 'off');
set(handles.pushbutton14, 'Enable', 'off');
set(handles.pushbutton4, 'Enable', 'off');
set(handles.pushbutton9, 'Enable', 'on');
set(handles.pushbutton3, 'Enable', 'off');
```

```
set(handles.pushbutton1, 'Enable', 'on');
set(handles.pushbutton6, 'Enable', 'off');
set(handles.pushbutton8, 'Enable', 'off');

% Reverse Direction --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'r');
set(handles.pushbutton4, 'Enable', 'off');
set(handles.pushbutton14, 'Enable', 'on');

% Increase Speed --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'i');

% Decrease Speed --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'d');

% Reset Path Plot --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global obj1;
cla(handles.axes4);
% grid(handles.axes4,'on');
% xlabel(handles.axes4, 'X Position (in)');
% ylabel(handles.axes4, 'Y Position (in)');
fprintf(obj1,'z');

% Turn Left --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
global obj1;
fprintf(obj1,'l');
set(handles.pushbutton14, 'Enable', 'on');
set(handles.pushbutton4, 'Enable', 'on');

% Turn Right --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'g');
set(handles.pushbutton14, 'Enable', 'on');
set(handles.pushbutton4, 'Enable', 'on');

% Straight --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;
fprintf(obj1,'s');
set(handles.pushbutton4, 'Enable', 'on');
set(handles.pushbutton14, 'Enable', 'off');

% Reset Plot --- Executes on button press in pushbutton21.
function pushbutton21_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton21 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.pushbutton23, 'Enable', 'off');
set(handles.pushbutton26, 'Enable', 'on');
set(handles.pushbutton21, 'Enable', 'off');

cla(handles.axes3);

% Stop Data Acquisition --- Executes on button press in pushbutton23.
function pushbutton23_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton23 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global var;
global pressure;
global t;
global x_position;
global y_position;
global voltage;
global proximity;
stop(var.tmr); % stops the timer
```

```
assignin('base','pressure',pressure);
assignin('base','t',t);
assignin('base','x_position',x_position);
assignin('base','y_position',y_position);
assignin('base','voltage',voltage);
assignin('base','proximity',proximity);
set(handles.pushbutton23, 'Enable', 'off');
set(handles.pushbutton26, 'Enable', 'off');
set(handles.pushbutton25, 'Enable', 'on');
set(handles.pushbutton21, 'Enable', 'on');

% Connect to Robot --- Executes on button press in pushbutton24.
function pushbutton24_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton24 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;

% Create the Bluetooth connection object if it does not exist
% otherwise use the object that was found.
h = waitbar(0,'Please wait...');
if isempty(obj1)
    obj1 = Bluetooth('Wall   RobotSmirf', 1);
else
    fclose(obj1);
    obj1 = obj1(1);
end

waitbar(0.5);

% Connect to instrument object, obj1.
fopen(obj1);

status = isvalid(obj1);

if status == 1
    set(handles.text6, 'BackgroundColor', 'green');
else
    set(handles.text6, 'BackgroundColor', 'red');
end

waitbar(1);
close(h);

set(handles.pushbutton24, 'Enable', 'off');
set(handles.pushbutton25, 'Enable', 'on');
set(handles.pushbutton23, 'Enable', 'off');
set(handles.pushbutton26, 'Enable', 'on');
set(handles.pushbutton21, 'Enable', 'off');
set(handles.pushbutton12, 'Enable', 'off');
```

```
set(handles.pushbutton13, 'Enable', 'off');
set(handles.pushbutton14, 'Enable', 'off');
set(handles.pushbutton4, 'Enable', 'off');
set(handles.pushbutton9, 'Enable', 'off');
set(handles.pushbutton3, 'Enable', 'off');
set(handles.pushbutton1, 'Enable', 'on');
set(handles.pushbutton6, 'Enable', 'off');
set(handles.pushbutton8, 'Enable', 'off');

% Reset Connection --- Executes on button press in pushbutton25.
function pushbutton25_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton25 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global obj1;

% Disconnect all objects.
fclose(obj1);

set(handles.text6, 'BackgroundColor', 'red');

pause(1);

% Connect to instrument object, obj1.
fopen(obj1);

status = isvalid(obj1);

if status == 1
    set(handles.text6, 'BackgroundColor', 'green');
else
    set(handles.text6, 'BackgroundColor', 'red');
end

% Begin Data Acquisition --- Executes on button press in pushbutton26.
function pushbutton26_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton26 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.pushbutton23, 'Enable', 'on');
set(handles.pushbutton26, 'Enable', 'off');
set(handles.pushbutton25, 'Enable', 'off');
set(handles.pushbutton21, 'Enable', 'off');
global var;
global i;
global obj1;
global t;
global pressure;
global x_position;
```

```
global y_position;
global voltage;
global proximity;
if strcmp(get(var.tmr, 'Running'), 'off')
    fprintf(obj1,'z');
    t = [];
    t = zeros(10000,1);
    pressure = [];
    pressure = zeros(10000,8);
    x_position = [];
    x_position = zeros(10000,1);
    y_position = [];
    y_position = zeros(10000,1);
    voltage = [];
    voltage = zeros(10000,1);
    proximity = [];
    proximity = zeros(10000,1);
    i = 1;
    var.x = 0;
    start(var.tmr);
end
fprintf(obj1,'a');
A = fscanf(obj1);
%A = query(obj1, 'a', '%c', '%s');
B = str2num(A);
pressure(i,1:8) = B(1,1:8);
t(i,:) = var.x;
plot(handles.axes3,t(1:i,:), pressure(1:i,:));
xlabel(handles.axes3,'Time (seconds)');
ylabel(handles.axes3,'Pressure (kPa)');
grid(handles.axes3,'on');

set(handles.text9,'String',B(1,9));

x_position(i,:) = B(1,10);
y_position(i,:) = B(1,11);
plot(handles.axes4,x_position(1:i,:), y_position(1:i,:));
xlabel(handles.axes4, 'X Position (in)');
ylabel(handles.axes4, 'Y Position (in)');
grid(handles.axes4,'on');

if B(1,12) > 1
    proximity(i,:) = B(1,12);
    set(handles.text10,'String','> 1');
else
    proximity(i,:) = B(1,12);
    set(handles.text10,'String',proximity(i,:));
end

var.x = var.x + 0.25;
i = i + 1;
```

```
% Keyboard buttons --- Executes on key press with focus on figure1 or any of its↙
controls.
function figure1_WindowKeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)
%display(eventdata.Key)
switch eventdata.Key;
     case 'uparrow'
        % set focus to the button
        uicontrol(handles.pushbutton6);
        % call the callback
        pushbutton6_Callback(handles.pushbutton6,[],handles);
     case 'downarrow'
        % set focus to the button
        uicontrol(handles.pushbutton8);
        % call the callback
        pushbutton8_Callback(handles.pushbutton8,[],handles);
     case 'leftarrow'
        % set focus to the button
        uicontrol(handles.pushbutton12);
        % call the callback
        pushbutton12_Callback(handles.pushbutton12,[],handles);
     case 'rightarrow'
        % set focus to the button
        uicontrol(handles.pushbutton13);
        % call the callback
        pushbutton13_Callback(handles.pushbutton13,[],handles);
     case 'return'
        % set focus to the button
        uicontrol(handles.pushbutton1);
        % call the callback
        pushbutton1_Callback(handles.pushbutton1,[],handles);
     case 'shift'
        % set focus to the button
        uicontrol(handles.pushbutton4);
        % call the callback
        pushbutton4_Callback(handles.pushbutton4,[],handles);
     case 'control'
        % set focus to the button
        uicontrol(handles.pushbutton3);
        % call the callback
        pushbutton3_Callback(handles.pushbutton3,[],handles);
     case 'slash'
        % set focus to the button
        uicontrol(handles.pushbutton14);
        % call the callback
```

```
        pushbutton14_Callback(handles.pushbutton14,[],handles);
end

% Save All --- Executes when user selects Save All Menu Tab
function Save_All_Callback(hObject, eventdata, handles)
% hObject    handle to Save_All (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

global var;
stop(var.tmr); % stops the timer
 % delete the timer object

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

F. TESTING DATA AND CALCULATIONS

Encoder Test Plan

Two encoder tests should be performed in order to determine the accuracy of the encoder count and position and speed calculations with the effects of tread slip during operation. The linear and angular motion tests obtain the theoretical speed and position values from the software that should match the actual measured motion values if the encoders are running properly. The steps taken to carry out these two tests are below.

Linear Motion Test

1. Zero the encoder counts and positions.
2. Run the robot forward then stop it when it has traveled 5 feet. Record the amount of time it takes the robot to travel that distance.
3. Record the encoder count, speed, position values from the serial terminal.
4. Repeat steps 1 through 3 four more times.

Angular Motion Test

1. Zero the encoder counts and positions.
2. Run the robot in a right turn then stop it when it has turned 90 degrees. Record the amount of time it takes the robot to span the 90 degree turn.
3. Record the encoder count, speed, position values from the serial terminal.
4. Repeat steps 1 and 2 four more times.

Pressure Sensor Test Equations Used

$$Pressure\ Sensor\ Calibration\ Calculations$$

$$Transfer\ Function\ from\ datasheet^{[18]}: V_{out} = V_S(0.009P + 0.04)$$

$$where\ V_{out}\ is\ the\ sensor\ voltage\ output, V_S\ is\ the\ sensor\ excitation\ voltage,$$

$$and\ P\ is\ the\ corresponding\ pressure\ reading\ in\ kPa$$

$$Solving\ for\ P: P = \left(\frac{V_{out}}{0.009V_S} - \frac{0.04}{0.009}\right) + offset$$

$$where\ a\ positive\ offset\ would\ decrease\ the\ output\ pressure$$

$$and\ a\ negative\ offset\ would\ increase\ the\ output\ pressure$$

Proximity Sensor Test Theoretical Models

The predicted results for the output of these proximity sensors were mainly based on the voltage versus distance regression models detailed by Malheiros et al.[26]. The performance of these sensors was tested in a variety of different ways and a non-linear regression model, shown in Figure 26 and tabulated in Table 11, for the output voltage as a function of the object distance was derived from the experimental data. The results were also compared to the predicted sensor output behavior provided in the sensor datasheet[19]. This behavior, seen in Figure 12, helped verify that the two sensor outputs and the theoretical model were behaving as the sensor was designed.

$$d = k_1\left(\frac{1}{v}\right)^2 + k_2\left(\frac{1}{v}\right) + k_3$$

In which k_1 has a value of 0.0346, k_2 has the value 0.251 and k_3 the value -0.0121. This function allows determining the distance measure from the output voltage. Its inverse is:

$$v = \frac{-k_2 - \sqrt{4k_1 d + k_2^{\,2} - 4k_1 k_3}}{2(-d + k_3)}$$

Figure 26: Models of output voltage as a function of distance for the proximity sensors

Table 11: Expected output voltages using the theoretical model for the proximity Sensor

Distance (m)	Output Voltage (V)
0.10	2.369
0.20	1.308
0.30	0.924
0.40	0.725
0.50	0.602
0.60	0.519
0.70	0.458
0.80	0.412

Proximity Sensor Test Plan

Several tests were ran in order to effectively determine the range, accuracy, and repeatability of the chosen proximity sensors. For the testing, the sampling rate that was used was 1 Hertz (one sample per second), which was slower than in normal operation. However, because all of the testing that was done was static testing, meaning that the sensor outputs were not being recorded while the objects were moving, a slow sampling rate was sufficient. To coarsely

negate the effects of noise in the static output signals of the sensors, the sensor outputs were measured and recorded for at least 30 seconds for each test, then the values obtained for each sensor were averaged to obtain a mean result for that test. Below are the steps that were taken to carry out each of these three tests. The experimental test setups for each test can be seen in Figures 27 – 30.

Range Test

1. Place object of selected width and height 40 centimeters in front of sensors.
2. Record the static output voltages in the serial terminal for at least 30 seconds.
3. Place object 15 degrees to the left and in front of the sensors, moving object along a radius of curvature of 40 centimeters. Face object forward, not angled toward sensors.
4. Record the static output voltages in the serial terminal for at least 30 seconds.
5. Repeat steps 3 and 4, increasing the angle from the sensors with each movement until the object is located 45 degrees from the front of the sensors.
6. Repeat steps 3 through 5 at angles to the right of the sensors.

Accuracy Test

1. Place object with selected surface reflectivity 40 centimeters in front of sensors.
2. Record the static output voltages in the serial terminal for at least 30 seconds.
3. Shine a light into the sensors without obstructing the sensors' view of the object.
4. Record the static output voltages in the serial terminal for at least 30 seconds.
5. Repeat steps 1 through 4 for same object with altered surface reflectivity values.

Repeatability Test

1. Place desired object 80 centimeters in front of sensors.
2. Record the static output voltages in the serial terminal for at least 30 seconds.
3. Move the object 10 centimeters closer to the sensors.
4. Record the static output voltages in the serial terminal for at least 30 seconds.
5. Repeat steps 3 and 4 until the object is 10 centimeters away from the sensors.
6. Move the object 10 centimeters farther away from the sensors.
7. Record the static output voltages in the serial terminal for at least 30 seconds.
8. Repeat steps 6 and 7 until the object is 80 centimeters away from the sensors.

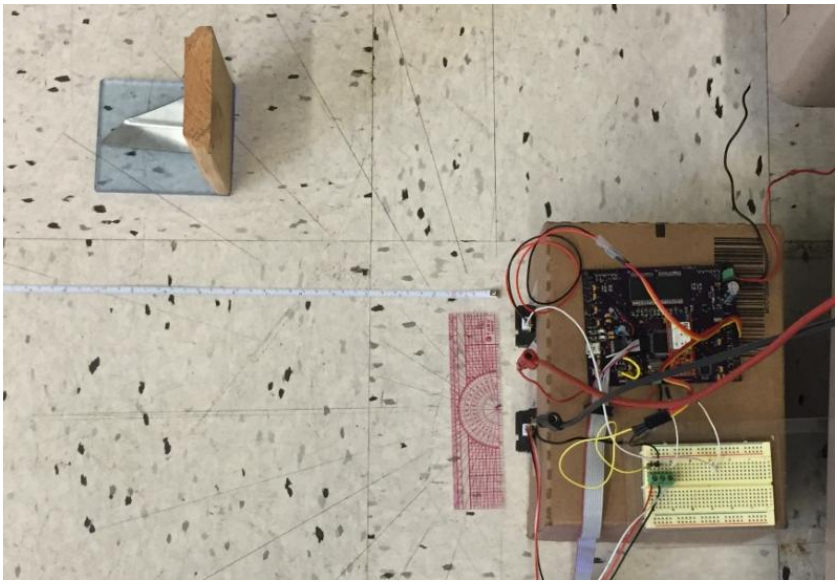

Figure 27: Experimental setup for range test using an object of small width and height located at an angle to the left of the proximity sensors

Figure 28: Experimental setup for accuracy test using an object with decreased surface reflectivity at a 40 centimeter distance in front of proximity sensors

Figure 29: Experimental setup for accuracy test using an object with increased surface reflectivity at a 40 centimeter distance in front of proximity sensors

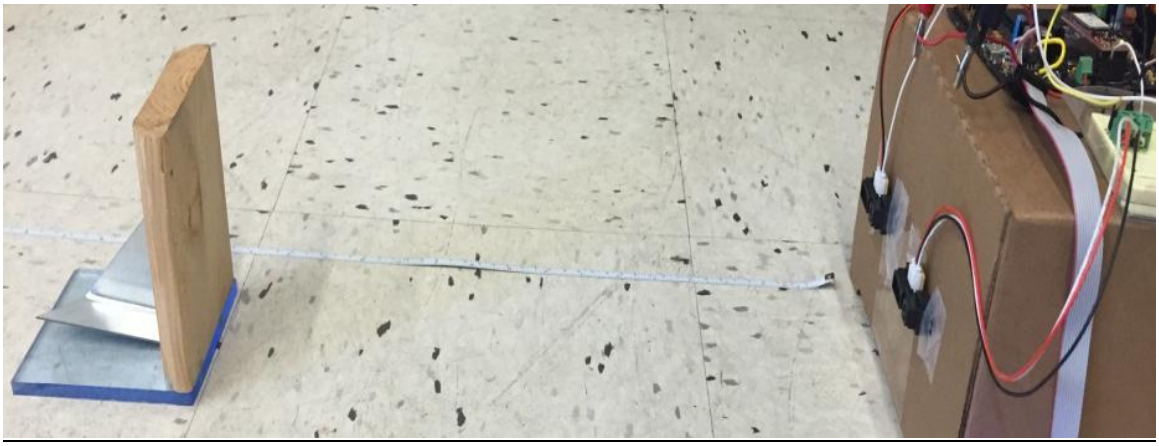

Figure 30: Experimental setup for repeatability test using object in front of the sensors

Proximity Sensor Test Results

It was desired to determine how changes in surface reflectivity of the object, object size, and the addition of a direct light source into the sensor affected the sensor outputs. In order to do this, numerous two-sample hypothesis tests were calculated. For this statistical analysis, several pairs of test condition combinations were directly compared to each other using a two-sided analysis to determine if the mean values read in the two tests could be considered equivalent. The null hypothesis for these calculations was that the two means being compared could be considered equal, while the alternative hypothesis was that the two means were not equal. The first pair of test means that were compared were the means from the left and right sensors for each of the accuracy test, range test, and repeatability test conditions. The equations used for these calculations can be found on Page 138. The tabulated results for each test can be seen in Tables 12 – 18.

Table 12: Average output voltages for left and right proximity sensors for range test

	Output Voltage (Volts)	
	Right Sensor	**Left Sensor**
Bookend - Straight	0.517	0.805
Bookend - 15° Left	0.176	0.046
Bookend - 30° Left	0.174	0.146
Bookend - 45° Left	0.178	0.148
Bookend - 15° Right	0.496	0.216
Bookend - 30° Right	0.239	0.185
Bookend - 45° Right	0.227	0.185
Book - Straight	0.451	0.471
Book - 15° Left	0.248	0.532
Book - 30° Left	0.226	0.185
Book - 45° Left	0.224	0.187
Book - 15° Right	0.928	0.485
Book - 30° Right	0.293	0.186
Book - 45° Right	0.226	0.183

Table 13: Average output voltages for left and right proximity sensors for accuracy test

	Output Voltage (Volts)	
	Right Sensor	**Left Sensor**
Book Regular - No Light	0.628	0.667
Book Regular - Light	0.634	0.664
Book w/ Foil - No Light	0.621	0.631
Book w/Foil - Light	0.624	0.630
Book w/Towel - No Light	0.634	0.750
Book w/Towel - Light	0.685	0.751

Table 14: Average output voltages for both proximity sensors in repeatability test

	Output Voltage (Volts)	
Distance (cm)	**Right Sensor**	**Left Sensor**
10	2.102	2.387
20	1.047	1.315
30	0.603	0.954
40	0.517	0.805
50	0.441	0.720
60	0.383	0.618
70	0.326	0.513
80	0.311	0.431

Table 15: Tabulated two-sample hypothesis testing values for comparison between left and right sensor means for each test condition in the accuracy test

	Variance		Samples		t_o	t_table
	Right	**Left**	**Right**	**Left**		
Book Regular - No Light	0.000219	0.000737	110	110	524.302	1.980
Book Regular - Light	0.000126	0.000915	113	113	350.559	1.980
Book w/ Foil - No Light	0.000229	0.000565	81	81	147.596	1.980
Book w/Foil - Light	0.000223	0.000579	133	133	117.506	1.980
Book w/Towel - No Light	0.000138	0.001179	186	186	1339.034	1.979
Book w/Towel - Light	0.000185	0.001105	116	116	635.586	1.980

Table 16: Tabulated two-sample hypothesis testing values for comparison between left and right sensor means for each test condition in the range test

	Variance		Samples		t_o	t_table
	Right	Left	Right	Left		
Bookend - Straight	0.000177	0.000813	66	66	2806.322	1.980
Bookend - 15° Left	0.000282	0.000501	69	69	-1875.385	1.980
Bookend - 30° Left	0.000260	0.001659	61	61	-128.300	1.980
Bookend - 45° Left	0.000301	0.001563	68	68	-158.747	1.980
Bookend - 15° Right	0.000269	0.001961	61	61	-1104.343	1.980
Bookend - 30° Right	0.000328	0.001965	63	63	-216.454	1.980
Bookend - 45° Right	0.000225	0.002142	63	63	-153.599	1.980
Book - Straight	0.000022	0.000139	67	67	1147.762	1.980
Book - 15° Left	0.000456	0.000006	33	33	3576.734	1.999
Book - 30° Left	0.000309	0.001902	37	37	-130.183	1.996
Book - 45° Left	0.000196	0.002174	31	31	-94.182	2.000
Book - 15° Right	0.000029	0.009553	37	37	-282.368	1.996
Book - 30° Right	0.000424	0.002413	29	29	-235.958	2.004
Book - 45° Right	0.000197	0.001504	38	38	-171.555	1.995

Table 17: Tabulated two-sample hypothesis testing values for comparison between left and right sensor means for each test condition in the repeatability test

	Variance		Samples		t_o	t_table
Distance (cm)	**Right**	**Left**	**Right**	**Left**		
10	0.000019	0.000008	70	70	114533.577	1.980
20	0.000011	0.000011	53	53	128310.327	1.985
30	0.000139	0.000458	38	38	4529.622	1.995
40	0.000177	0.000813	66	66	2806.322	1.980
50	0.000232	0.000721	30	30	2017.982	2.002
60	0.000260	0.001021	30	30	1224.098	2.002
70	0.000243	0.000611	30	30	1557.469	2.002
80	0.000142	0.000996	34	34	697.348	1.998

Table 18: Tabulated two-sample hypothesis testing values for light and no light source sensor means for each test condition and each sensor in the accuracy test

	t_o right	t_o left	t_table right	t_table left
Book Regular	3.899	-1.594	1.980	1.980
Book w/ Foil	0.211	-0.175	1.980	1.980
Book w/Towel	-2.525	0.544	1.980	1.980

Using a 5% significance level, the calculated t-statistic value was either higher than the positive t-statistic value from the table or lower than the negative t-statistic value from the table, which corresponds to the rejection of the null hypothesis. Therefore, the sensors did not produce the same results for the same test conditions. The other pair of test means that were compared were the means for each sensor with and without an additional light source. The findings showed that the effect of the light source was difficult to determine. For higher surface reflectivity, the addition of light did not cause the null hypothesis to be rejected. This correlated to no presumption being made against the null hypothesis. However, for lower reflectivities, the right sensor rejected the null hypothesis, while the left sensor did not.

Upon analyzing the mean values produced by the sensors in each test condition, it was observed that the surface reflectivity of the object being sensed did not greatly affect the sensor data. In the accuracy test, the change in the mean sensor outputs for the varying reflectivity values was minimal. This suggested that the surface reflectivity of the object being sensed did not affect the sensor readings. This would correlate with the datasheet graph, seen in Figure 12, which plotted the sensor output for two significantly different surface reflectivities and produced practically identical outputs for both cases.

Additionally, during the range test, it was determined that the orientation of the sensor affected the sensor performance. These sensors contained infrared emitters and receivers. When the receiver was on the outside edge of the test area, the sensor was able to sense the object at a wider angle. However, when the emitter was on the outside edge, the sensor lost sight of the object at a smaller angle position. When the object was also located right in front of the emitter, the object was less likely to reflect any infrared light to the receiver, causing the sensor readings to go below the no object detected readings.

It was also observed that the field of view of the sensor was relatively limited. During the range test, both objects usually stopped being seen by the sensors at around 45 degrees at the distance of 40 centimeters away. However, this field of view would be dependent on the size of the object being sensed and how well the object reflected infrared signal back to the receiver. Additional testing would need to be performed with various objects to determine how these factors, along with which direction the object was facing, would affect the overall field of view.

Another important observation was that smaller objects located in front of the proximity sensors generally resulted in smaller output values than the larger objects. This was most likely because the smaller objects did not have as much surface area as the larger objects and were unable to reflect as much infrared light back to the receivers. However, the alignment of the objects with the front of the sensors may have impacted these results. If, for some reason, the smaller object was more centered and closer to perpendicular with the sensors than the larger object was, the smaller object would produce larger readings, and vice versa.

Proximity Sensor Test Equations Used

$$Two - Sample\ Hypothesis\ Testing - Two\ Sided\ Calculations$$

$Null\ Hypothesis: \mu_1 = \mu_2$

$Alternative\ Hypothesis: \mu_1 \neq \mu_2$

$The\ null\ hypothesis\ can\ be\ rejected\ if\ t_0 > t_{\alpha/2,DOF}\ or\ t_0 < -t_{\alpha/2,DOF}$

$where\ t_{\alpha/2,DOF}\ is\ the\ t\ statistic\ (t\ distribution\ table)\ and\ \alpha\ is\ the\ significance\ level$

$$Degree\ of\ Freedom\ (DOF) = n_1 + n_2 - 2$$

$$Average\ Sensor\ Reading = \frac{\sum_{i=1}^{n} y_i}{n} = \bar{y} \quad where\ y_i\ are\ the\ samples\ for\ n\ samples$$

$$Variance = \frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n-1} = S^2$$

$$Common\ Variance = \frac{(n_1-1)S_1^2 + (n_2-1)S_2^2}{n_1+n_2-2} = S_P^2 \quad where\ S_1^2\ and\ S_2^2\ are\ sample\ variances$$

$$t\ statistic\ for\ test = \frac{\bar{y}_1 - \bar{y}_2}{S_P\sqrt{\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} = t_0$$

$$P - value = 2(\alpha) \quad where\ \alpha\ is\ found\ using\ DOF\ and\ t_0$$

G. BILL OF MATERIALS

Table 19: Bill of materials with all components purchased for completion of project

Part Name	Qty.	Unit Price	Total
ATMega1281 Microcontroller	1	$12.20	$12.20
VNH5019ATR-E Motor Drivers	2	$8.75	$17.50
1 kΩ Resistor	10	$0.10	$1.00
10 kΩ Resistor	2	$0.20	$0.20
1.5 kΩ Resistor	2	$0.16	$0.32
330 Ω Resistor	4	$0.15	$0.60
470 Ω Resistor	2	$0.10	$0.20
1 µF Capacitor	3	$0.44	$1.32
0.1 µF Capacitor	19	$0.33	$6.27
22 pF Capacitor	2	$0.25	$0.50
10 µF Capacitor	1	$0.45	$0.45
1N4942 Diode	2	$0.46	$0.92
MOSFET	1	$0.82	$0.82
LM340MP-05 Voltage Regulator	2	$1.31	$2.62
COM-00536 16 MHz Clock Crystal	1	$0.95	$0.95
Green LED's	1	$4.95	$4.95
Red LED's	1	$2.95	$2.95
FT232RL USB UART IC	1	$4.50	$4.50
Ferrite Bead	1	$0.28	$0.28
16 Channel Multiplexer	1	$0.95	$0.95
Breakaway Male Headers	1	$1.50	$1.50
2 Pin Screw Terminals	2	$0.75	$1.50
6 Pin Female Header	1	$0.50	$0.50
USB Port Connector	1	$1.50	$1.50
2 x 3 Male Header	1	$0.50	$0.50
BlueSMiRF Silver Modem	1	$24.95	$24.95
Pocket AVR Programmer	1	$14.95	$14.95
Printed Circuit Board	1	$67.00	$67.00
6" Jumper Wire Pack	9	$1.55	$13.95
Male/Female Deans Connector	2	$0.95	$1.90
25' – 16 Gauge Machine Tool Wire	2	$5.00	$10.00
Test Lead Banana Plug Adapter	4	$2.56	$10.24
Molon 24 Volt 50 in-lb. Motor	2	$53.16	$106.32
Sharp IR Distance Sensor	2	$14.95	$29.90
Hamlin Gear Tooth Sensor	2	$20.48	$40.96
MPX5100DP Pressure Sensor	8	$16.09	$128.72
8mm Bore Hardened Steel Gear	2	$19.99	$39.98
		TOTAL	$553.87

H. OPERATION MANUAL

Circuit Board Programming Instructions

Required Software:

- Notepad++, Kate, or similar code editing environment
- Command Window
- WinAVR
- Zadig

Getting Started

Before being able to program the ATMega1281 microcontroller on the circuit board, all of the supporting software and development tools need to be downloaded and installed. These instructions are detailed for Windows computers, but similar procedures apply when using other machines as well. To begin programming, WinAVR, an open source software development tool that supports AVR programming, needs to be downloaded and installed on the Windows computer. WinAVR can be downloaded from the WinAVR website. This set of tools provides the ability to compile and install all of the appropriate files onto the microcontroller using Command Window line commands. Once downloaded, open the command window and change the working directory to the folder in which all of the project code is located in. In order to do this, the following commands need to be used:

- *dir* – View available folders in current directory
- *cd* FolderName – Change current directory to FolderName
- *cd..* – Make parent folder current directory

In order to use the Pocket AVR Programmer, AVR driver installation is first required for Windows machines. For other machines, all of the drivers should automatically install when the programmer is plugged into the computer. Zadig is a tool that can be used to install all the necessary drivers to the computer and the Zadig tool and USBtiny drivers can be downloaded from the SparkFun Pocket AVR Programmer Hookup Guide found on the SparkFun website. Once the tool is downloaded, plug the programmer into the computer via USB and run the "zadig.exe" file, which will open up a window as shown in Figure 31. Select the AVR device, most likely called "Unknown Device #1" and the only option available. Click the arrows on the driver selection bar (seen to the right of the large green arrow) until "libusb-win32 (vX.X.X.X)" appears, then click "Install Driver". After the installation process is complete, a message saying "The driver was installed successfully" should appear on the screen. If not, these drivers can be installed manually as well. The instructions for manual installation can be found on the SparkFun Pocket AVR Programmer Hookup Guide website.

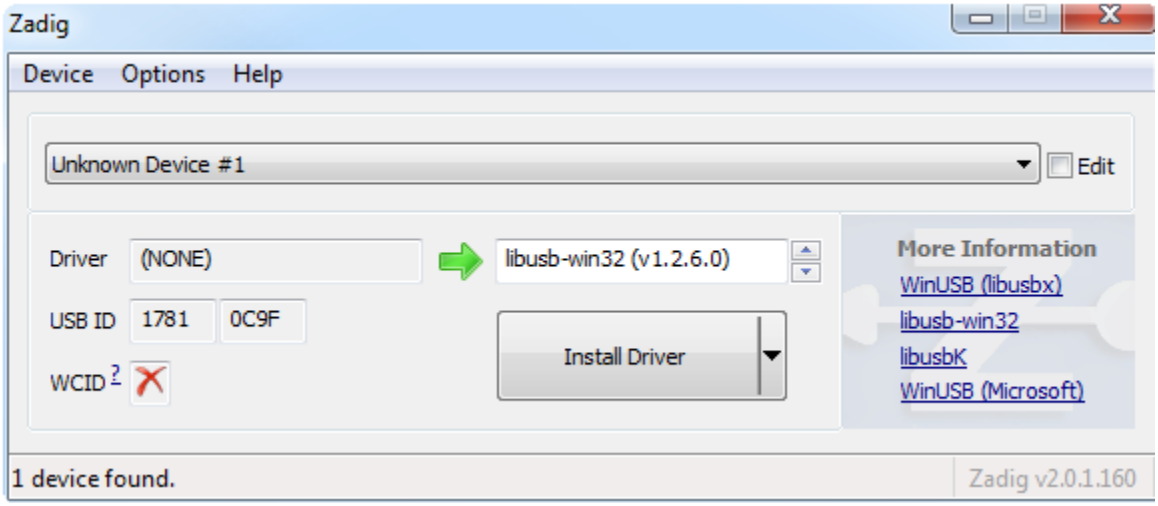

Figure 31: Zadig tool window used to install Pocket AVR Programmer

Programming the Board

Once all of the necessary software and development tools are installed, the board can be programmed. First, a check to make sure that the saved code does not have any formatting errors in it must be performed. This is done by inputting the "make clean all" command into the Command Window. Once this command has been executed, the "make install" command can be input into the Command window to initiate the programming of the board.

I. ADAMS MOTOR MODEL WORK

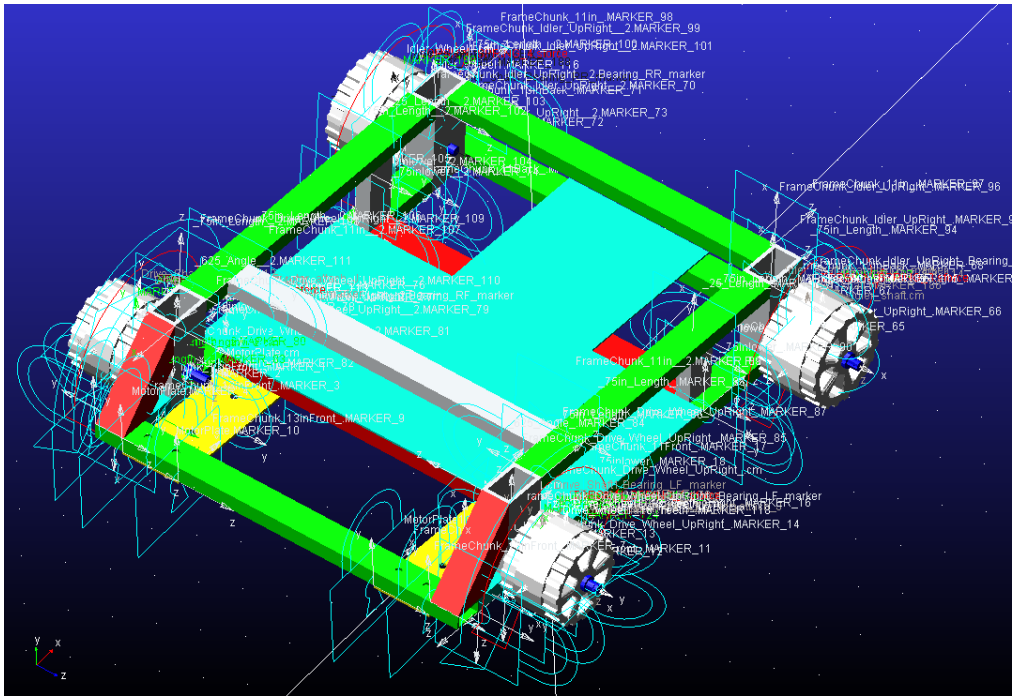

Figure 32: Full ADAMS motor model created for robot base frame and drive system

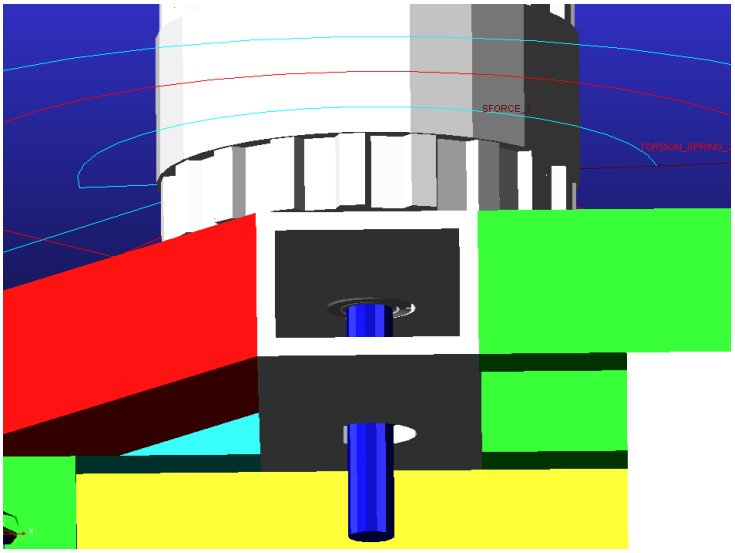

Figure 33: ADAMS bearing model implementation on front right drive shaft and frame upright housing

J. SIMULINK MOTOR CONTROLLER WORK

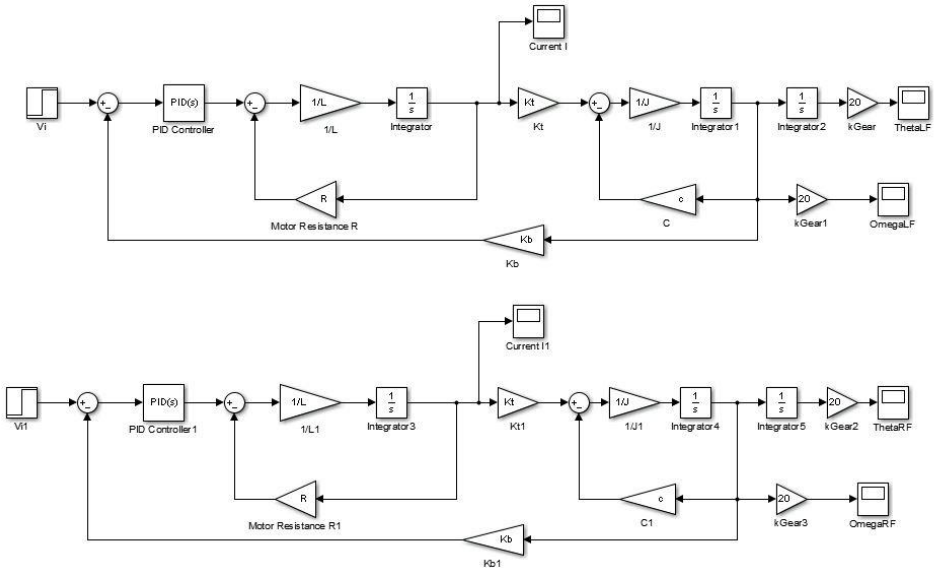

Figure 34: Simulink PID closed loop controller for two DC motors with theoretical motor plant model

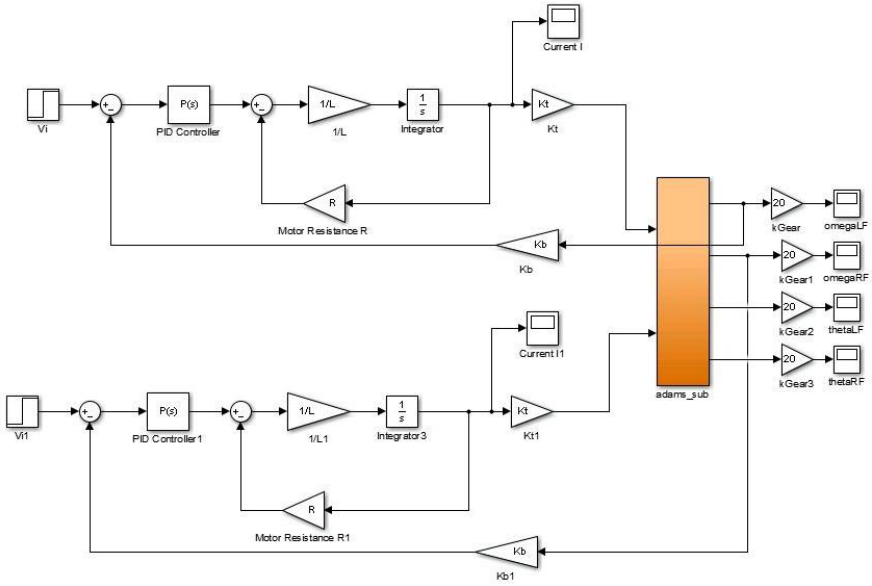

Figure 35: Simulink PID closed loop controller for two DC motors with ADAMS plant implemented