

Spring 5-2016

# Modular Design of an Educational Robotics Platform

Zhen Wei

*Rose-Hulman Institute of Technology*

Follow this and additional works at: [http://scholar.rose-hulman.edu/electrical\\_grad\\_theses](http://scholar.rose-hulman.edu/electrical_grad_theses)



Part of the [Electrical and Electronics Commons](#)

---

## Recommended Citation

Wei, Zhen, "Modular Design of an Educational Robotics Platform" (2016). *Graduate Theses - Electrical and Computer Engineering*. Paper 7.

This Thesis is brought to you for free and open access by the Graduate Theses at Rose-Hulman Scholar. It has been accepted for inclusion in Graduate Theses - Electrical and Computer Engineering by an authorized administrator of Rose-Hulman Scholar. For more information, please contact [bernier@rose-hulman.edu](mailto:bernier@rose-hulman.edu).

# **Modular Design of an Educational Robotics Platform**

A Thesis

Submitted to the Faculty

of

Rose-Hulman Institute of Technology

by

Zhen Wei

In Partial Fulfillment of the Requirements for the Degree

of

Master of Science in Electrical Engineering

May 2016

© 2016 Zhen Wei



**ROSE-HULMAN INSTITUTE OF TECHNOLOGY**

**Final Examination Report**

Zhen Wei

Name

Electrical Engineering

Graduate Major

Thesis Title Modular Design of an Educational Robotics Platform

**DATE OF EXAM:**

**EXAMINATION COMMITTEE:**

	<b>Thesis Advisory Committee</b>	<b>Department</b>
Thesis Advisor:	<b>Carlotta Berry</b>	<b>ECE</b>
	<b>Mark Yoder</b>	<b>ECE</b>
	<b>David Fisher</b>	<b>ME</b>

**PASSED**     X    

**FAILED**

## **ABSTRACT**

Wei, Zhen

M.S.E.E

Rose-Hulman Institute of Technology

May 2016

Modular Design of an Educational Robotics Platform

Thesis Advisor: Dr. Carlotta Berry

The goal of this thesis is to design a modular educational robotics platform to improve the limitation of current educational robotics platforms, such as limited pins, single programming language, and single programming device. This platform uses an SPI bus for modularity and to solve the problem of limited pins on current educational robot platforms. A Raspberry Pi, which runs a 32-bit Embedded Linux System, has been used to build the central control for this educational robotics platform to enable it to use different programming languages and to be programmed by different devices. The modules and libraries for stepper motors and IR sensors have been built for this robot, and the example projects, basic control, obstacle avoidance, and wall following, show that this educational robotics platform can be used as a platform for basic artificial intelligence design. This thesis also shows how to design a custom module, which enables users to design their own modules and put them into their robot projects.

## ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and help of many individuals who gave their valuable assistance in preparation and completion of my projects.

First, thanks to my thesis advisor, Dr. Carlotta A. Berry, for her inspiration and encouragement and her steadfast support.

Second, thanks to Jack Shrader and Gary Meyer, for their support with technology and help in implementing the robot.

Third, thanks to my committee members, Dr. Mark Yoder and Dr. David S. Fisher, for their suggestions to improve my work.

Finally, thanks to my family and friends for their support and for giving me the strength to complete this thesis.

## TABLE OF CONTENTS

ABSTRACT	2
ACKNOWLEDGEMENTS	3
TABLE OF CONTENTS	4
LIST OF FIGURES	7
LIST OF TABLES	9
1. Introduction	10
2. Literature Review	12
3. Design of the Modular Robotic Platform	15
3.1 Mechanical Design	15
3.2 Electrical Design	16
3.2.1 Communication Interface	16
3.2.2 Central Controller and Custom Module	18
3.2.3 Motor Control Module	22
3.2.4 IR Sensor Module	24
3.2.5 Power Supply	26

3.2.6 Conclusion	27
3.3 Firmware Design for Modules	30
3.3.1 SPI Communication	30
3.3.2 Stepper Motor Control	33
3.3.3 IR Sensor Control	39
3.4 Central Controller Setup	42
3.5 Firmware Design for Central Controller	46
4. Demo Project	48
5. Comparison	52
6. Conclusion and Future Work	55
LIST OF REFERENCES	57
APPENDICES	58
APPENDIX A: PROJECT PARTS LIST	58
APPENDIX B: MSP430G2553 FIRMWARE CODE	61
APPENDIX C: RASPBERRY PI FIRMWARE CODE	76
APPENDIX D: EXAMPLE PROJECT CODE	86

Demo Project 1: Basic Control	86
Demo Project 2: Obstacle Avoidance	89
Demo Project 3: Wall Following	95
APPENDIX E: LIST OF DEMO VIDEOS	101
APPENDIX F: USER'S MANUAL	102
APPENDIX G: SIMPLE EDUCATIONAL PROJECT	116



## LIST OF FIGURES

Figure 1: Overall View of the Platform	16
Figure 2: SPI Connection for Modular Educational Robotics Platform	18
Figure 3: Image of Custom Module	20
Figure 4: The Schematic of Custom Module	21
Figure 5: PCB Layout of Custom Module	22
Figure 6: Schematic of Motor Controlling Module (H-Bridges)	23
Figure 7: Schematic of Motor Controlling Module (Microcontroller and Connectors)	24
Figure 8: Schematic of IR Sensor Module (Voltage Divider)	25
Figure 9: Schematic of IR Sensor Module (Microcontroller and Connectors)	26
Figure 10: Schematic of Voltage Regulator Circuit	26
Figure 11: Power Wiring Diagram	27
Figure 12: Schematic of Base Board	28
Figure 13: PCB Layout of Base Board	29
Figure 14: Image of Base Board	29
Figure 15: USCI Control Register 0	31

Figure 16: Flow Chart for SPI communication	32
Figure 17: Excitation Sequences for Different Drive Modes	33
Figure 18: Bipolar Wound Stepper Motor	33
Figure 19: Full Step Control Signal Generated by Time Delays	35
Figure 20: Full Step Control Signal Generated by Timer Interrupts	36
Figure 21: Timer_A Control Register	37
Figure 22: TACCTLx, Capture/Compare Control Register	38
Figure 23: Part of ADC10 Control Register 1	39
Figure 24: ADC10AE0, Analog (Input) Enable Control Register 0	40
Figure 25: ADC10MEM, Conversion-Memory Register, Binary Format	40
Figure 26: ADC10CTL0, ADC10 Control Register 0	40
Figure 27: Flow Chart for a Basic Function Design	47
Figure 28: Flow Chart for Obstacle Avoid Project	50
Figure 29: Flow Chart for Wall Following	51

## LIST OF TABLES

Table 1: Used Pin Connections for Raspberry Pi 2	19
Table 2: Control Command for Basic Control Demo	48

## 1. Introduction

Most robots or microcontrollers in the market have limited I/O ports, ADCs, DACs, and communication ports. For example, the Android Robot, which is currently used in ECE 425 Mobile Robot class, only has 8 extra I/O ports, and the Traxster II has 5 analog input ports and 9 digital I/O ports. Therefore, this may not be enough for users to develop their own additional robot features. It would be more economical and time effective to use a modular or “plug and play” robot when developers wish to use many different sensors, motors, or other technologies. It may be argued that C is the universal robot programming language, but it may no longer be the most popular language among users. The developer may prefer to program in Python, JAVA, Lua, or some other language of choice, so it would preferable to have flexibility in the software used and sometimes it even makes it easier to implement the robot algorithms by using a programming language other than C. However, this is currently not the case, as most robot platforms are still based on C programming. Also, the PC is not the only tool developers use for robot control and programming; smartphones, tablets, and notebooks have become more popular, as these are cheaper, simpler and more easily accessible devices. This thesis will focus on finding the solution to the problems current educational robotics platforms have, including limited pins, single programming language, and single programming device. Also, the robot should be able to do all the projects which the robot used in the ECE 425 Mobile Robotics class can do.

This research designed a modular educational robotics platform, which was built on one central controller that functions as the command center or brain for several different modules. It

is like a “plug and play” robotic platform. The modules in this research include IR sensors, motors, and a custom module. Because any kind of microcontroller will have limited IO ports, this research used SPI bus communication technology to compensate for this limitation. The central controller is the “master” and the modules are the “slave”. The master and slaves are connected using the 4 wire SPI mode. Normally the central controller will set the sensor modules selection pin to ON to select the sensor module and make sure they can send the data back to the central controller. This way, only one module can communicate with the central controller at a time. After the central controller gets data from sensor modules, it may send some control commands to the other modules. Then the central controller turns off the current sensor module selection pin, and turns on the new selection pin.

A 32-bit Embedded Linux System was run on a central controller, which enables it to use different programming languages and to be controlled by different devices. The 32-bit Embedded Linux System is as same as a normal Linux System, and it can run all kinds of programming languages such as C, Java, Python, Lua, and so on. Therefore, users can program on this robotics platform just like a normal Linux System. The Linux System allows the robot to be connected, controlled, and communicated through Secure Shell (SSH). The SSH is a cryptographic (encrypted) network protocol allowing remote login and other network services to operate securely over an unsecured network. By using the SSH, users can remote login to this robotics platform and control, program this robotics platform through the network by any kind of devices. Therefore, the SSH allows this platform be programed by different devices, such as Android phones or tablets, Apple devices, Windows PCs, or Chromebooks.



## 2. Literature Review

In Andrey Shvartsman, Maurice Tedder, and Chan-Jin Chung's *A Modular Mobile Robotic Platform As An Educational Tool In Computer Science And Engineering*, they designed a mobile robotic platform which can be used as a tool for computer science and engineering education. This platform consists of several integrated modules, including a laptop computer that serves as the main control module, a microcontroller-based motion control module, a vision-processing module, a sensor interface module, and a navigation module. They concluded that their modular mobile robot platform design is inexpensive due to off the shelf components and a mass-production manufacturing model. In this robot, the main control module controls the others primarily through serial ports and USB hubs. Also, this robot can be controlled by multiple programming languages, as the main software development environment is setup on a laptop computer [1]. Their design uses a laptop computer as the main control module, which also can be understand as a central controller, which makes the robot platform expensive, but it improved the idea of building a flexible software development environment on a robot platform. The design for this thesis follows this idea and uses an embedded Linux development board as a central controller, which gives the flexible software development environment and also decreases cost.

In *Raspberry Pi Learning Resources* there is a robot called Robobulter which is powered by Raspberry Pi with Raspbian Linux OS. The Robobulter used the GPIOs to control the motors directly. Python has been used to program this robot [2]. For this robot, it is easy to program the robot, and gives user the flexible software development environment. However, Raspberry Pi has a limited number of GPIO pins, and by using the GPIO to directly control the robot, it quickly uses up the GPIO. In this thesis, the robot has be designed into modular, which means it use the

Raspberry Pi to control all the modules and the module control the robot. In this way, the user can add any features into the robot by adding a module and do not need to worry about how many GPIOs Raspberry Pi has.

S Piperidis, et. al. presented a paper to reduce the cost of a mobile robot for education and research, while maintaining its capabilities. The low cost modular mobile robot ALE worked well for both education and research, because it has been used for a course and the research lab in the past two years. In this design, the Bluetooth communication module has been used to make the robot modular, and it is controlled by a personal computer through a serial port in real time [3]. For real time control, the robot platform designed in this thesis uses Wifi instead of a serial connection, so that the robot can be controlled by different types of devices running different operating systems and wirelessly. This makes the robot platform more user friendly. The modular design of ALE is good, but using Bluetooth to transfer data between modules wirelessly is cumbersome, because the module still need wired power and all of the modules are very close to each other. Additionally the Bluetooth chip will increase the cost.

In Josep M. Mirats Tur and Carlos F. Pfeiffer's *Mobile Robot Design in Education*, they described the course on robot design that electronics systems engineers take in their last semester at the Instituto Tecnológico de Estudios Superiores de Monterrey (ITESM), Monterrey, Mexico Campus. In this course, the use of Project Oriented Learning and collaborative learning are proposed. They then describe the design and implementation of a modular, low-cost, three-wheeled autonomous robotic platform to serve as a base platform from which different applications, educational and research, could be mounted. For this robot platform, all control hardware is modular, distributed, and interconnected using a CAN (controller area network) bus,



which has been proven to be a robust solution for industrial applications [4]. For a modular design, using a CAN bus is excellent. However, most cheap microcontrollers are not designed with a CAN port, which means if a CAN bus is used to connect these microcontrollers, they require an external CAN controller and CAN transcoder. To add additional chips increases the cost and power consumption. The design in this thesis uses a SPI bus instead of a CAN bus, because SPI is one of the most popular protocols in microcontrollers and most microcontrollers have at least one internal SPI communication port.

### **3. Design of the Modular Robotic Platform**

#### 3.1 Mechanical Design

When designing a new mobile robot, the wheeled mobile robot classification is always chosen first. The book, *Mobile Robotics for Multidisciplinary Study*, mentions that there are four prevalent wheeled mobile robot classifications distinguished by the arrangement of the driving and steering wheels. They are differential drive, synchronous drive, tricycle drive and car drive [5]. The most common type of wheeled mobile robot is differential drive, which is simple for programming and locomotion. For designing an educational robot, simplicity is one of the most important factors. Also compared with other popular educational mobile robots, most of them are using differential drive wheeled mobile robot classifications. For example, all of the iRobot, Traxster, TraxBot, CEENBot, Arduino Robot, and Khepera used the differential drive classification. Therefore, the differential drive classification has been chosen for this platform. The design of the platform has two wheels driven independently on a common axis with one caster on the back for stability.

The platform design in this thesis uses a robot body from an old Traxster robot, which included the wheels and frame, and two stepper motors to drive the mobile robot. The biggest part of the mechanical design in this thesis is choosing the motors. Compared with the normal mobile robots currently in the market, most of them used DC motors with or without encoders. All of the Traxster, TraxBot, Arduino Robot, and Khepera use two DC motors with encoders to build their motion system. Only the CEENBot used stepper motors. In general, DC motors are connected by two wires and driven by a power signal, which means the location can be figured

out by differentiation overtime. There is a relative margin of error, meaning the exact location cannot be known. However, stepper motors are driven by position signals, which means it can be located with less error than dead reckoning. Therefore, mobile robots which are controlled by stepper motors can run much more accurately than those controlled by DC motor without encoder.

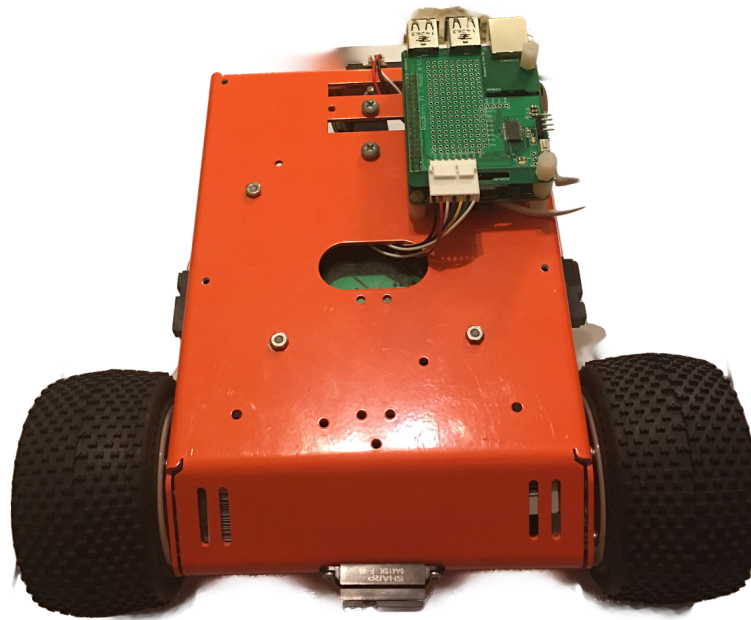


Figure 1: Overall View of the Platform

## 3.2 Electrical Design

### 3.2.1 Communication Interface

The electrical control circuit is the most important part of this platform design. Thinking about the features, this design requires a stepper motor control module, an IR sensor module, and a custom module. Before starting to design the modules, the communication interface must be designed.

For a modular design, the communication interface can be a wired bus or some wireless networks, so the data can be transferred between components in the system. Inside the robot, everything will be wired, so there is no reason to add a wireless communication module into each of these modules. Therefore, this design uses a wired communication bus to connect all of the modules together. The most popular wired bus communication interfaces are Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I<sup>2</sup>C), and Controller Area Network (CAN Bus). The SPI bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems and most of the microcontrollers have at least one SPI port. I<sup>2</sup>C is a multi-master, multi-slave, single-ended, serial computer bus. It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers, and most of the microcontrollers have an I<sup>2</sup>C port also. CAN Bus is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. It is a message-based protocol, designed originally for multiplexing electrical wiring within automobiles. For a modular mobile robot, the ideal would be CAN bus, but most low cost microcontroller do not have CAN bus port. If using CAN bus, an extra CAN bus module would be required for each module, so it would increase the cost of the robot. Comparing I<sup>2</sup>C and SPI, the transfer speed of SPI is faster than I<sup>2</sup>C. Normally, the speed of I<sup>2</sup>C is between 100 kHz to 400 kHz. However, SPI do not have upper limit speed in the protocol itself, it is only limited by its electrical interface and connected devices. Also, SPI support full-duplex communication but I<sup>2</sup>C does not support. Therefore, SPI has been chosen for this design, which will give the platform more potential expandability in the future, even though the transfer speed and full-duplex communication does not affect current design. The design of this robotics

platform is most like a one master and several slaves system, which is shown in Figure 2. The master controller is designed based on a Raspberry Pi 2, which is a 32-bit Embedded Linux platform, and called the Central Controller. All the other modules are designed based on the MSP430G2553 microcontroller. This thesis includes three module: motors control, IR sensor and custom.

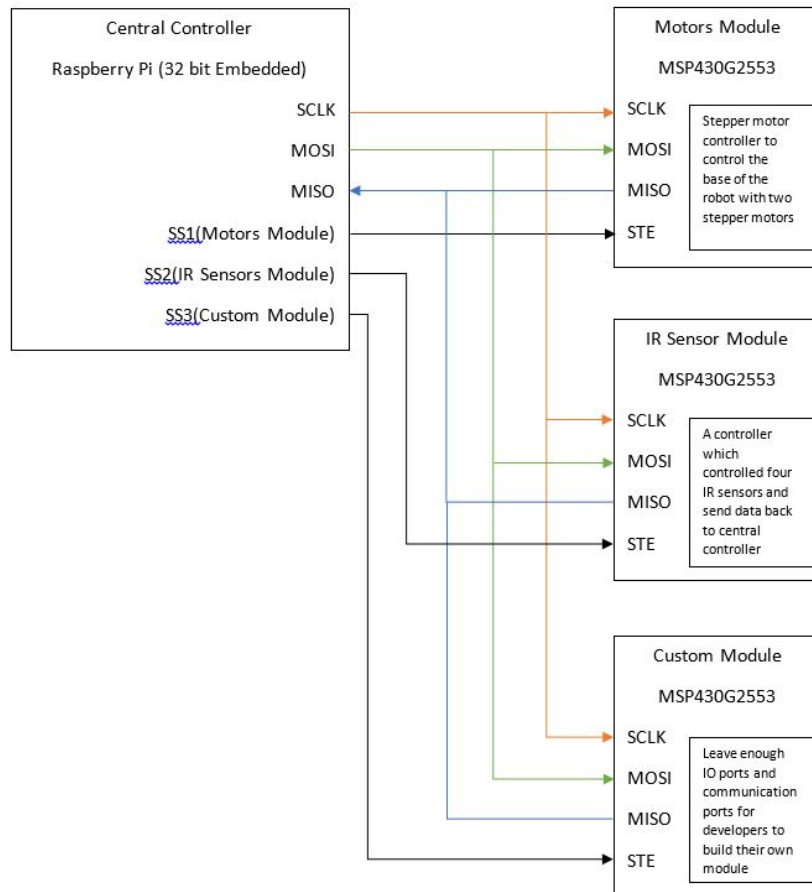


Figure 2: SPI Connection for Modular Educational Robotics Platform

### 3.2.2 Central Controller and Custom Module

The central controller is used to control everything and lets users program the robot. It is built with a Raspberry Pi 2 which runs the Debian Linux OS. Based on the GPIO pin out for the

Raspberry Pi 2, the SPI should be connected to the GPIO 10, GPIO 09, and GPIO 11. The other GPIO pins are used as the SPI slave transmit enable pin. The GPIO 04, the GPIO 17, and the GPIO 18 has been used as slave transmit enable pin in this project for motor control module, IR sensor module, and custom module. The pin connections are shown in Table 1. On the final design, all of these pins except slave transmit enable pin for custom module are wired into a 7 pin header on a custom module, which makes it easier to connect motor control module and IR sensor module into this platform. The image of custom module is shown in Figure 3.

**Table 1: Used Pin Connections for Raspberry Pi 2**

Connections	Pin #	Pin Name	Description
SPI Connections	19	GPIO 10	SPI Master Out Slave In
	21	GPIO 09	SPI Master In Slave Out
	23	GPIO 11	SPI Clock pin
SPI Slave Transmit Connections	07	GPIO 04	Slave Transmit Enable Pin for Motor Control Module
	11	GPIO 17	Slave Transmit Enable Pin for IR Sensor Module
	12	GPIO 18	Slave Transmit Enable Pin for Custom Module

The custom module is designed for user to add their own circuits into the robot platform. This module only has a MSP430G2553 chip in the circuit. The Adafruit part P2223 header connector, a normal 2 by 20 female header with extra long pins, has been used to connect the custom module and Raspberry Pi 2. The P2223 header will allow users to add more custom designed modules into the robot easily. Users just need to plug in their own modules using the P2223 header and change the GPIO pins for SPI slave transmit enable. The sample schematic of custom modules in this design is shown in Figure 4. J1 is raspberry pi 2 GPIO pin connectors,

which will use the P2223 header and connect to the Raspberry Pi. J1 also shows the pinout for SPI MISO (GPIO 10), MOSI (GPIO 09), CLK (GPIO 11), and all slave transmit enable pins for three different modules, motor control module (GPIO 17), IR sensor module (GPIO 04), and custom module (GPIO 18). J2 shows the connector with all SPI pins, and it is used to connect the motor control module and IR sensor module into the SPI bus. The schematic also shows the circuits design for the MSP430G2553, which include protection, programming, and resetting. This custom module also allows access to every free pins of the MSP430G2553 for users. The PCB layout is shown in Figure 5. On the PCB layout, there is a protoboard area, made of 0.1” pitch through holes, for users to solder their own circuits on the module. The PCB is designed following the shape of the Raspberry Pi 2 to make sure it can be plug in and mounted easily. The programming headers on the side of the PCB makes it easy for users to programming the module.

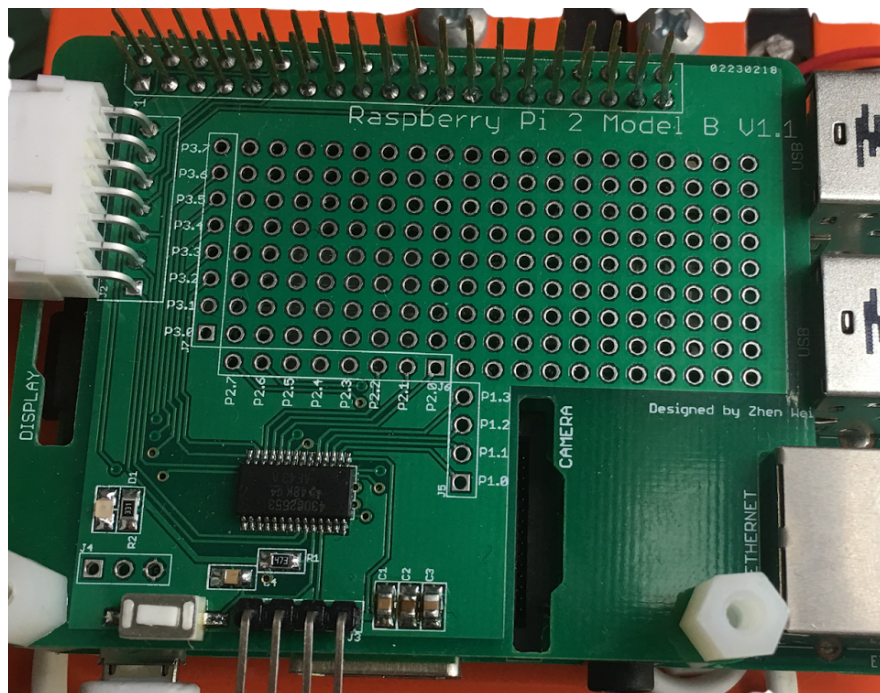


Figure 3: Image of Custom Module

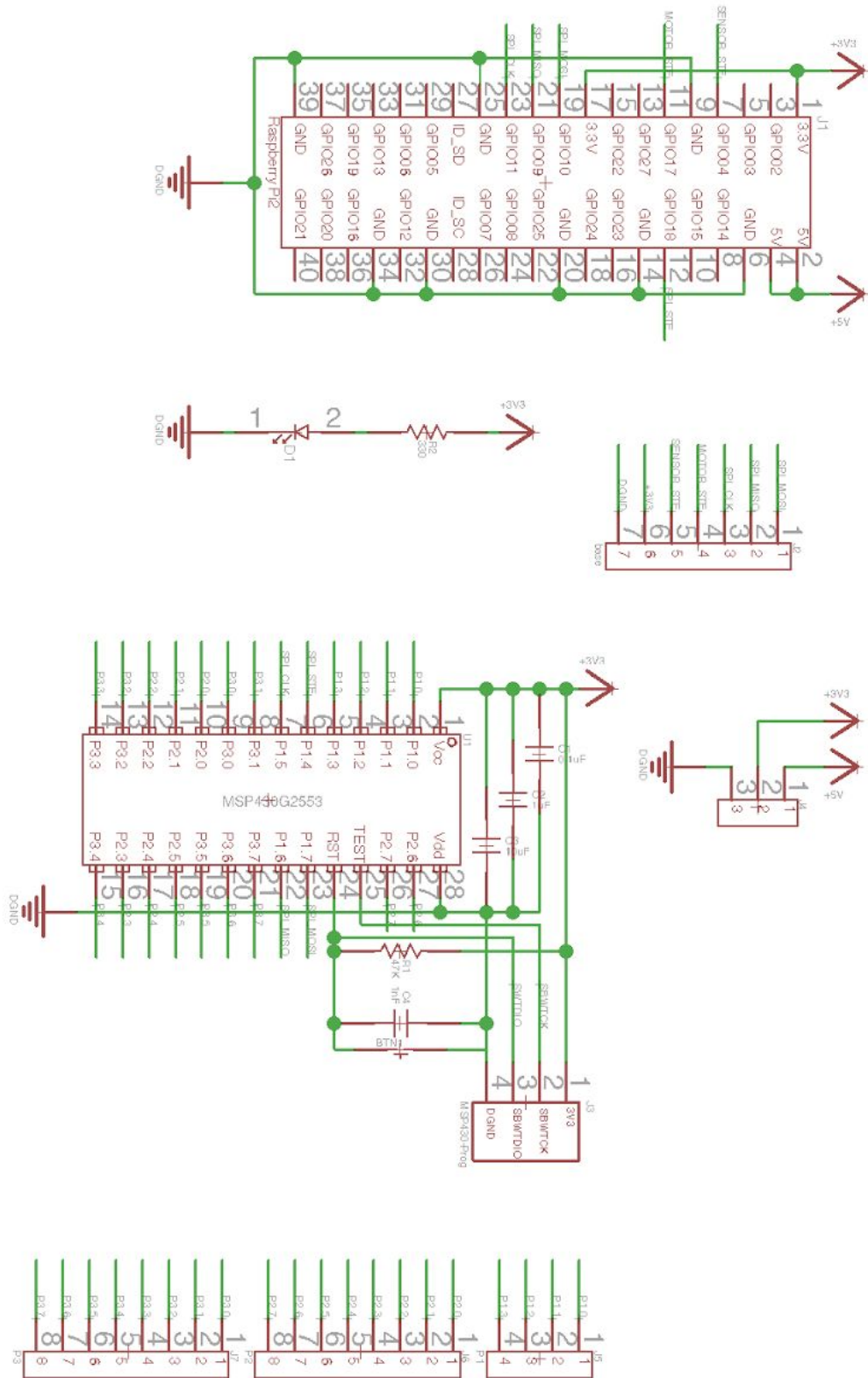


Figure 4: The Schematic of Custom Module



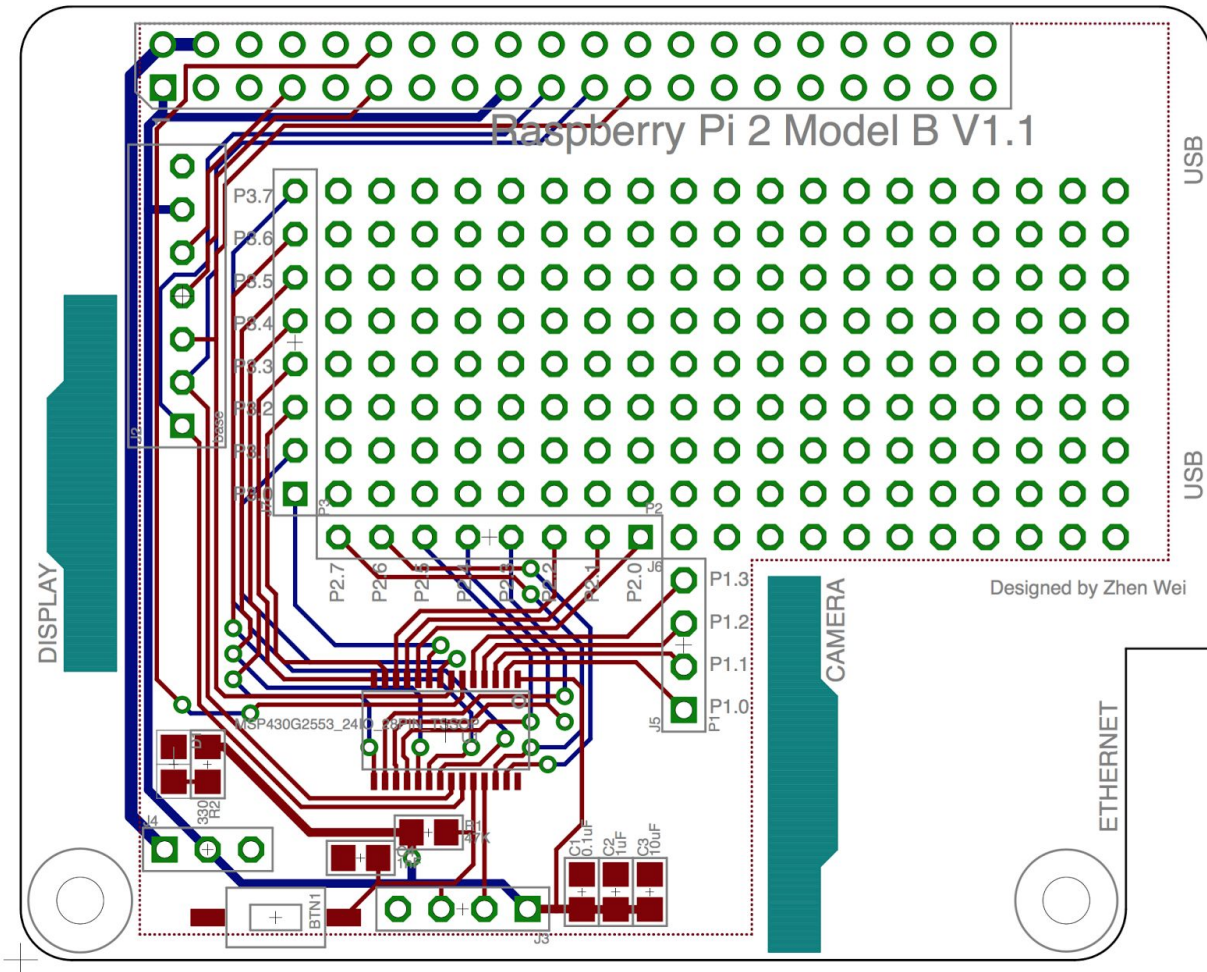


Figure 5: PCB Layout of Custom Module

### 3.2.3 Motor Control Module

The motor control module controls the motion of the robot. In this design, the motor control module is designed to control two stepper motors. The Adafruit stepper motor with product id 324 has been used. It is a bipolar stepper motor which is driven by 12V and a maximum current of 340 mA. This 4-wire bipolar stepper motor has 1.8 degrees per step and 200 steps per revolution. The red and yellow wires are paired for coil #1, and the green and grey wires are paired for coil #2. The Adafruit web also shows that the wires should be in order for

red, yellow, green, grey. The 4-wire bipolar stepper motor should be easily controlled by two full H-bridges. Each full H-bridge controls one of the coils. It can make the current go through the coil in either direction by turning the H-bridge on or off. Because the maximum current limit for this 324 Adafruit 4-wire bipolar stepper motor is 340 mA, the two full H-bridges chip L293DD who is made by STMicroelectronics with 600 mA current limit has been chosen. The schematic of the stepper control, which used two L293DDs, is shown in Figure 6.

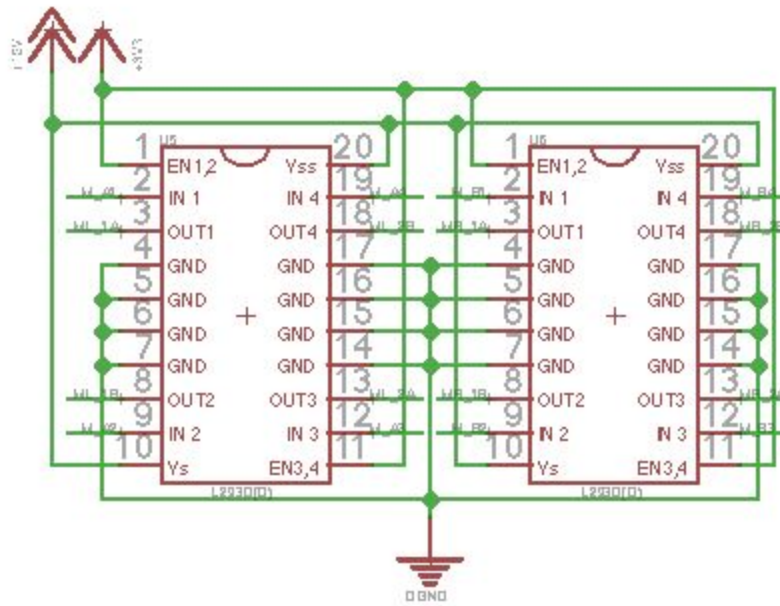


Figure 6: Schematic of Motor Controlling Module (H-Bridges)

The supply voltage  $V_s$  and logic supply voltage  $V_{ss}$  are connected to 12V, which requires 4.5V as minimum and 36V as maximum that shows in STMicroelectronics' datasheet for L293DD.

The minimum of the input high voltage level is 2.3V and the logic high for MSP430G2553 output is 3.3V, so the logic control pins of L293DD can connect directly to the MSP430G2553, and the EN pins connected to 3.3V. All the output pins are connected to the wires from the stepper motor in order of red, yellow, green, grey. The schematic of all the connection between H-bridges and microcontroller, and between H-bridges and motors are shown in Figure 7, and

the PCB Layout is shown in Figure 13. Due to the modular design, the final version puts the motor control module and IR sensor module on the same board which will make it easy to wire and hold on the base, the SPI jumpers are used to connect or disconnect the module from the SPI bus, which are shown in Figure 7 as J8 and J9.

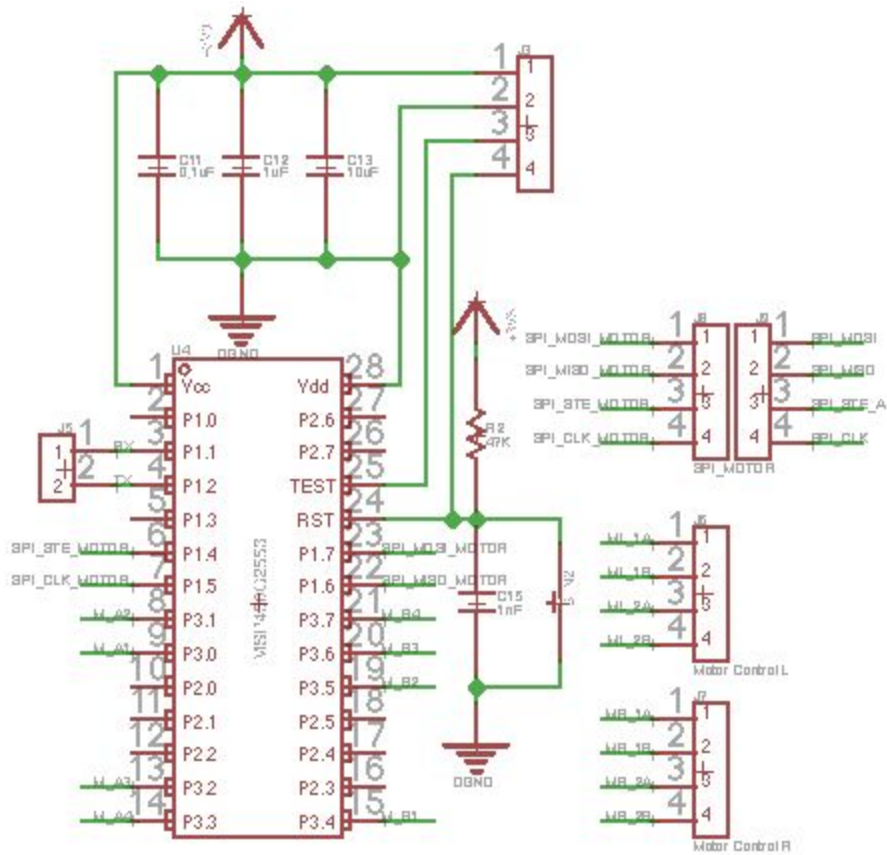


Figure 7: Schematic of Motor Controlling Module (Microcontroller and Connectors)

### 3.2.4 IR Sensor Module

The IR sensor module is used to get the data from four Sharp 0A41SK0F IR distance measuring sensor unit, which is composed of an integrated combination of position sensitive detector (PSD), infrared emitting diode (IR-LED), and signal processing circuit. Its operating supply voltage is 4.5V to 5.5V, and the analog out range is -0.3V to  $V_{cc}+0.3V$ . In this design,

the Vcc is 5.5V, so the maximum out voltage is 5.3V. The maximum ADC input voltage for MSP430G2553 only can be 3.3V, which means Vcc for MSP430G2553. Therefore, the output signal from the IR sensor cannot be connected to the MSP430G2553 directly. A voltage divider has been used here to reduce the output voltage, and as shown in Figure 8.



Figure 8: Schematic of IR Sensor Module (Voltage Divider)

The schematic shows the ratio of the voltage divider is  $\frac{1}{2}$  calculated by the following equation:

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

where both of the R1 and R2 are equal to 10KOhm. Since the maximum output value of the sensor is 5.3V, the maximum output value of the voltage divider is 2.75V, which is less than 3.3V, and it is safe to connect to MSP430G2553. Also it uses the same design as motor controlling module to connect and disconnect the IR sensor module to SPI bus. All the circuits are shown in the Figure 9.

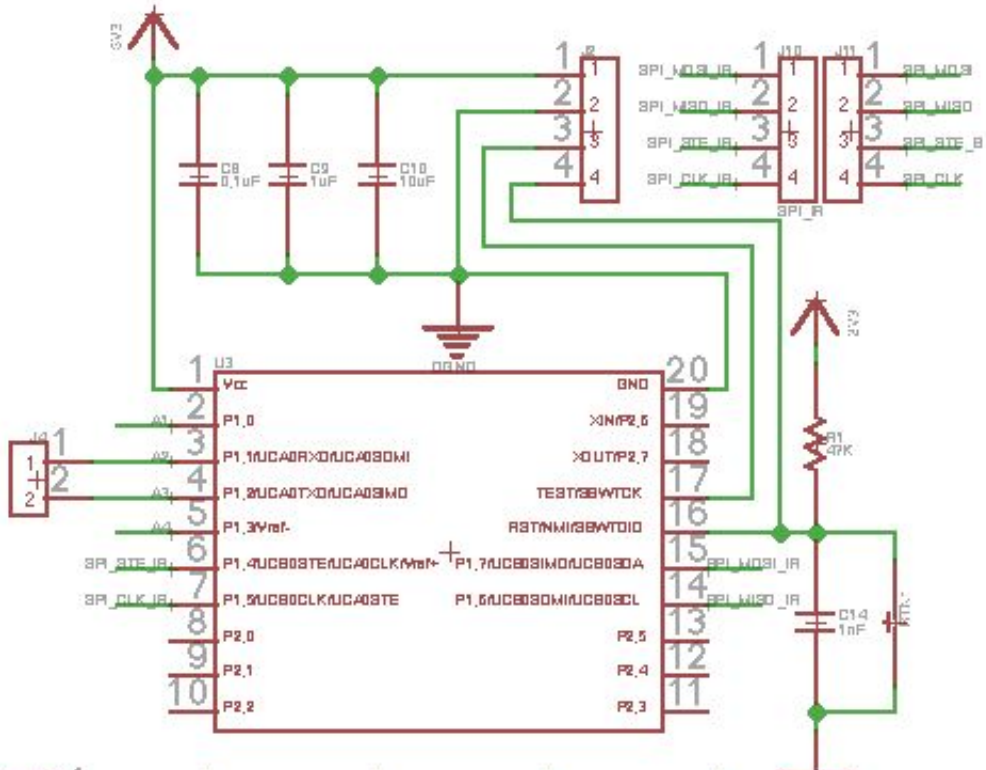


Figure 9: Schematic of IR Sensor Module (Microcontroller and Connectors)

### 3.2.5 Power Supply

The power supply is built with a power regulator circuit, an on/off switch, and a 12V Li-ion Battery. The power regulator circuit supports 2 different voltage levels, 5V and 3.3V. The circuit is shown in Figure 10.

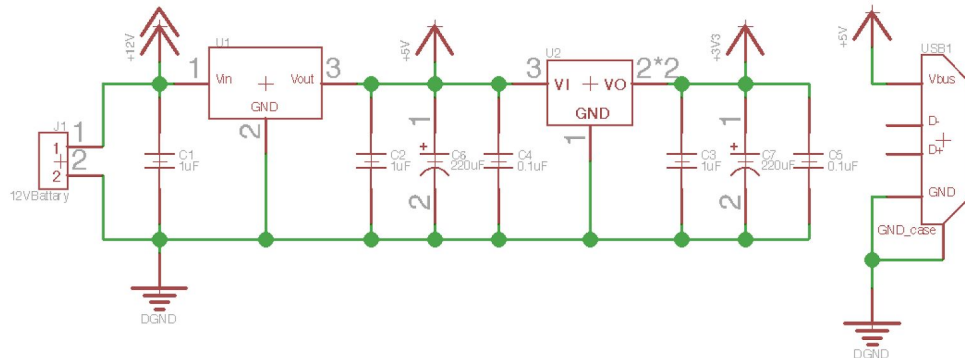


Figure 10: Schematic of Voltage Regulator Circuit

The U1 is a non-isolated switching regulator whose part number is V7805-2000, which supports 5V 2A with 12V input. It is used to support all the 5V parts, such as IR sensors, and Raspberry Pi 2. The second level of the power regulator is a linear regulator with 3.3V and 800mA output. The 3.3V voltage level supports all of the microcontroller and other 3.3V level circuits. It also provides a micro USB connector, which makes it easy to power the Raspberry Pi. The 12V Li-ion Battery connects to the voltage regulator circuit through a switch, which turns the robot on and off. The wiring diagram is shown in Figure 11.

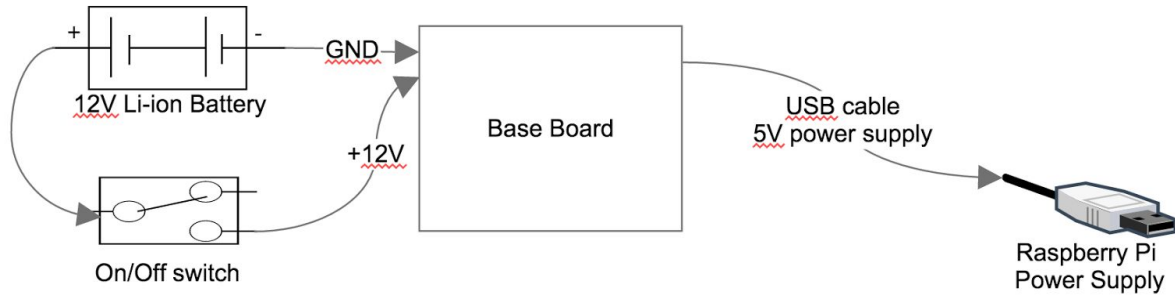


Figure 11: Power Wiring Diagram

On the base board, there is a micro USB connector which is used to supply the 5V power for Central Control. Also the 3.3V should be wired to all modules.

### 3.2.6 Conclusion

Since all of the Power systems, IR sensors, motors will be installed to the base, the design combines all of them together onto one board, which will be called as base board and shown in Figure 14. The base board includes the power regulators, IR sensor module, and motor control

module, and is installed to the bottom of the robot base frame. The final schematic design of the base board is shown in Figure 12, and the PCB layout design in Figure 13.

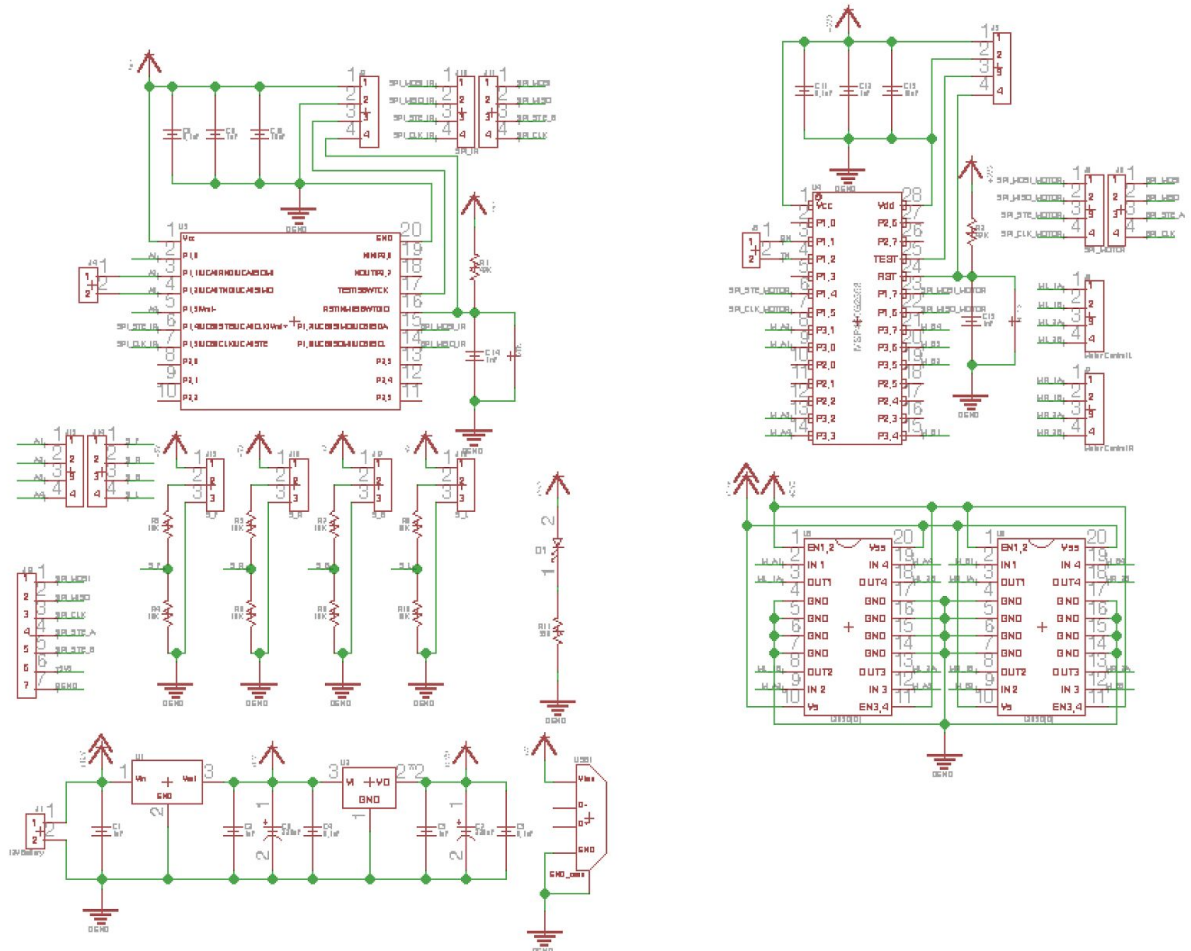


Figure 12: Schematic of Base Board

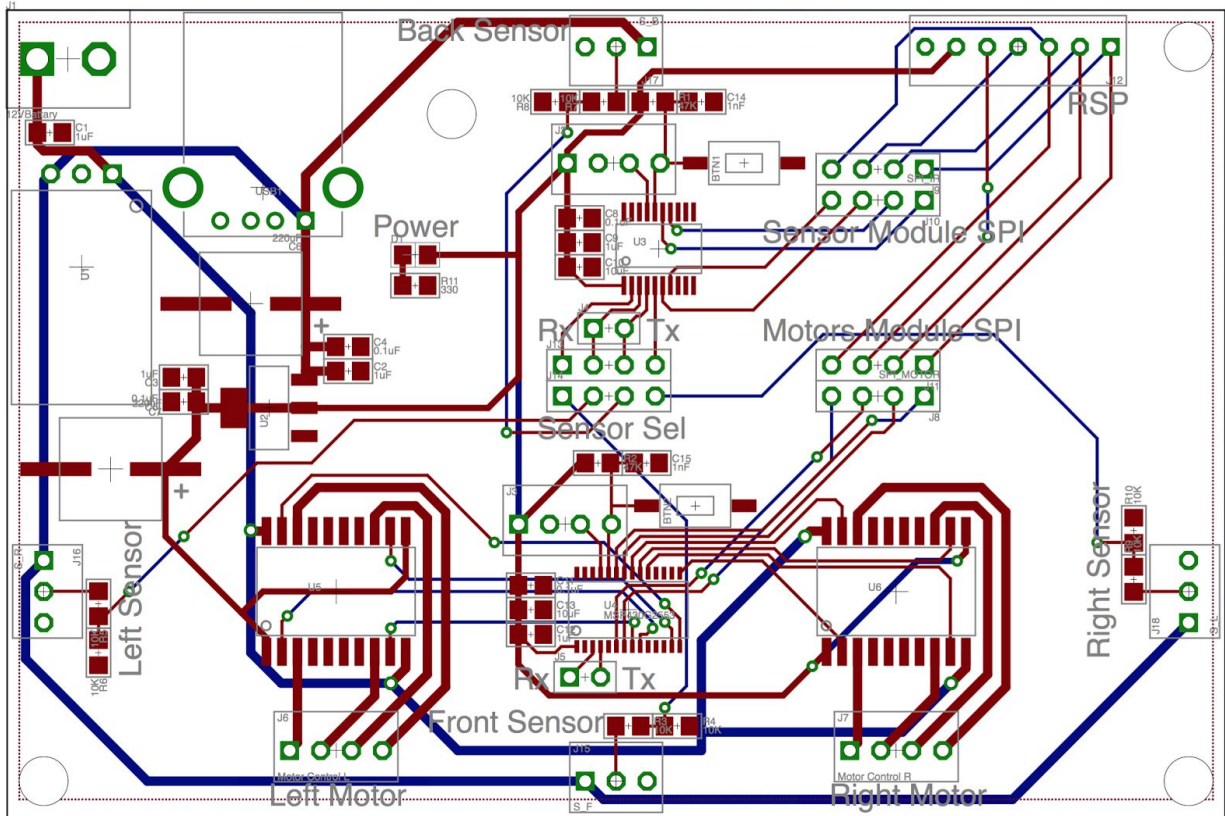


Figure 13: PCB Layout of Base Board

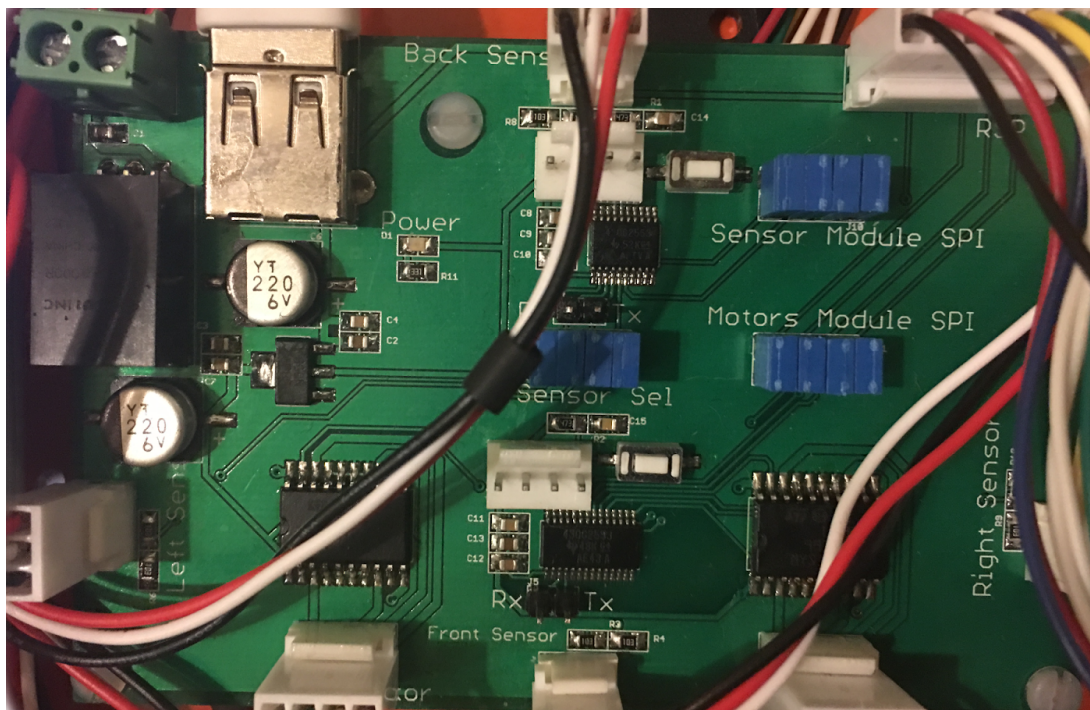


Figure 14: Image of Base Board



### 3.3 Firmware Design for Modules

As the words said, hardware is the body and the firmware is the soul [6]. A good robot design cannot only have hardware. The firmware is also a very important part of the design. The modules should have their own firmware to control the module and communicate with the central controller. The firmware design should include the SPI communication, stepper motor controller, and IR sensor control.

#### 3.3.1 SPI Communication

All of the modules in this design have been chosen as an SPI slave. The four-wire SPI bus has been decided to use in this project because the four-wire SPI bus can select the modules which it wants to communicate with by using the fourth wire. In MSP430G2553, there is a module called a universal serial communication interface (USCI) which supports multiple serial communication modes. For MSP430G2553, there are two USCI modules named USCI\_A and USCI\_B. Both USCI\_A and USCI\_B modules support SPI mode. The SPI mode of USCI module connects the MSP430G2553 to an external system via four pins: UCxSIMO (slave input, master output, which is output from master), UCxSOMI (slave output, master input, which is output from slave), UCxCLK (serial clock, which is output from master), and UCxSTE (slave transmit enable or slave select).

The design for modules select the SPI to 4-pin SPI operation with slave mode, and the data length will be 8-bit. For slave mode, the code needs to set UCMST to 0. The UC7BIT set to 0 to select 8-bit data length, and the UCMODE set to 1 to get 4-pin SPI with UCxSTE active

high. Finally, the USCI control register 0 should be set as follows: “UCB0CTL0 |= UCMSB + UCSYNC + UCCKPH + UCMODE\_1;”. This register is shown in Figure 15.

7	6	5	4	3	2	1	0
UCCKPH	UCCKPL	UCMSB	UC7BIT	UCMST	UCMODEx		UCSYNC=1
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	
<b>UCCKPH</b>	Bit 7	Clock phase select. 0 Data is changed on the first UCLK edge and captured on the following edge. 1 Data is captured on the first UCLK edge and changed on the following edge.					
<b>UCCKPL</b>	Bit 6	Clock polarity select. 0 The inactive state is low. 1 The inactive state is high.					
<b>UCMSB</b>	Bit 5	MSB first select. Controls the direction of the receive and transmit shift register. 0 LSB first 1 MSB first					
<b>UC7BIT</b>	Bit 4	Character length. Selects 7-bit or 8-bit character length. 0 8-bit data 1 7-bit data					
<b>UCMST</b>	Bit 3	Master mode select 0 Slave mode 1 Master mode					
<b>UCMODEx</b>	Bits 2-1	USCI mode. The UCMODEx bits select the synchronous mode when UCSYNC = 1. 00 3-pin SPI 01 4-pin SPI with UCxSTE active high: slave enabled when UCxSTE = 1 10 4-pin SPI with UCxSTE active low: slave enabled when UCxSTE = 0 11 I <sup>2</sup> C mode					
<b>UCSYNC</b>	Bit 0	Synchronous mode enable 0 Asynchronous mode 1 Synchronous mode					

Figure 15: USCI Control Register 0

As mentioned before, the SPI mode of USCI module has independent interrupt capability for the receive and transmit feature. The firmware uses the interrupt only to receive, which means any time one module of the robot gets a message from central controller, the SPI interrupt for receive will wake up the module from the low power mode, and it will send the message back. This method helps the robot save power. Inside the USCI interrupt, the program checks the flag first, the flag helps to identify what kind of interrupt is. If the receive flag is up, which means the chip is receiving data from an external device, it gets data from the receive buffer, and identifies what the data is. If the data is not an end of a command, it stores the data into a buffer

array, which is built by 10 8-bit values. When the code get the data end of the command, it will return the command, and execute the command.

To transmit a message back to the central controller, the slave will package the message and push it into a global char array tx\_buf. When communication starts, the slave pushes the data into a transmit buffer one bit at a time to get the data from the receive buffer. In the same way, the master gets the data from the buffer at the same time when it sends the data. In the firmware code, there are two functions that send the data back. One is spi\_putc() function, which is used to check transmit flag and push a 8-bit data into the transmit buffer when the transmission is free. Another one is return\_message, which helps to transmit a string with 10 8-bit datas. The SPI code spi\_slave.c and its head file spi\_slave.h are attached in APPENDIX B.

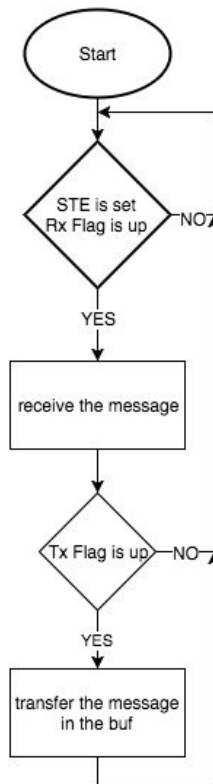


Figure 16: Flow Chart for SPI communication

### 3.3.2 Stepper Motor Control

To control a stepper motor, the most common driver modes are wave drive, full step drive, and half step drive [7]. The excitation sequences for those drive modes are summarized in Figure 17. The Figure 18 shows a bipolar stepper motor.

Phase	Wave Drive				Normal full step				Half-step drive								
	1	2	3	4	1	2	3	4	1	2	3	4	5	6	7	8	
A	•				•			•	•							•	•
B		•			•	•			•	•							
$\overline{A}$			•				•	•			•	•	•				
$\overline{B}$				•				•						•	•	•	

Figure 17: Excitation Sequences for Different Drive Modes [7]

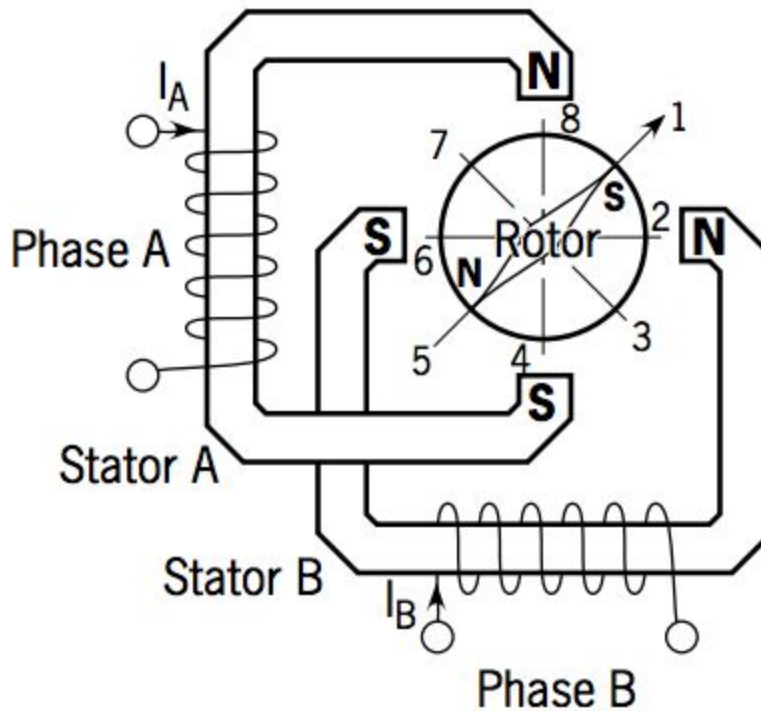


Figure 18: Bipolar Wound Stepper Motor [7]

In Wave Drive, as Figure 17 shows, it only energizes one winding at a time. For the stepper motor shown in Figure 18, it energizes the winding by following the sequence  $A \rightarrow B \rightarrow \overline{A} \rightarrow \overline{B}$ .

The rotor steps are from the position  $8 \rightarrow 2 \rightarrow 4 \rightarrow 6$ . In Full Step Drive, two phases are energized at any given time. Like Figure 17 shows, the sequence to energize the stator is

$AB \rightarrow \bar{A}\bar{B} \rightarrow \bar{A}B \rightarrow A\bar{B}$ , and the rotor steps from position  $1 \rightarrow 3 \rightarrow 5 \rightarrow 7$ . The Half Step Drive is

mixed with both wave and full step drive modes, as Figure 17 shows, only one winding energizing status will be changed at one time. The sequence of the stator is energized is

$AB \rightarrow B \rightarrow \bar{A}\bar{B} \rightarrow \bar{A} \rightarrow \bar{A}B \rightarrow \bar{B} \rightarrow A\bar{B} \rightarrow A$ , and the rotor steps from position

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8$ . [7] Full Step Drive is used in the design, due to full step drive mode provides improved torque and speed performance, which means compare with half step drive and wave drive, full step drive provides more torque than the other two.

To generate the control wave for full step drive, four GPIO out pins have been used to generate the output signal. The speed of stepper motors is controlled by the time difference between two steps, which means if the steps are changing faster, the motor will turn much faster. There are two ways to make the stepper motor control work: one is using a time delay to change the GPIO output values inside the main loop, and the another idea is using the timer interrupt to control the GPIO output values, and time difference to trigger the interrupt are used to control the speed. The first way to control the robot is easy to make it happen, but when adding more code into the main loop, the speed should be changed to be not that accurate. However, the second way can make the motor run more accurately, and it do not effect by how many codes need to be run inside the main loop.

The full step control signal for using a time delay between changing the GPIO output values shows in Figure 19. The graph also shows that it takes 24.07 ms for four steps. The code set the speed of the motor as 50 rpm, which means 6 ms per step and 24 ms for four steps.

However, in Figure 20, the full step control signal for using timer interrupts, shows that it only takes 23.98 ms to run four steps. Comparing these data, for every four steps, the controlling signal generated by time delay cost 0.07 ms more than the ideal value. The controlling signal generated by timer interrupt cost 0.02 ms less than the ideal data. The result shows that the signal generated by the timer interrupt is more accurate than the signal generated by time delay. Therefore, the timer interrupt has been used to control the stepper motors in this project to increase the accuracy.

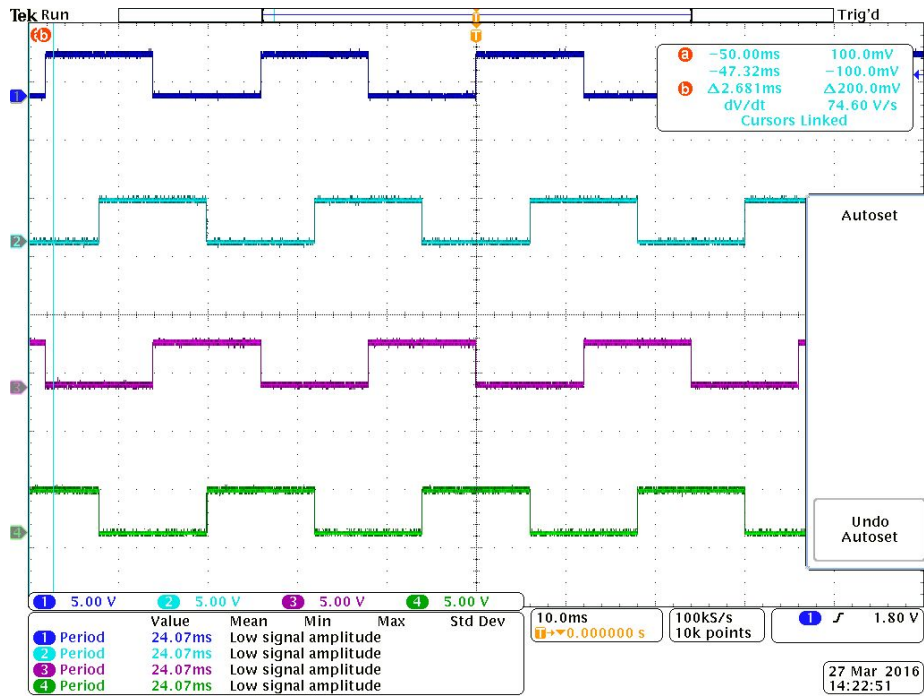


Figure 19: Full Step Control Signal Generated by Time Delays

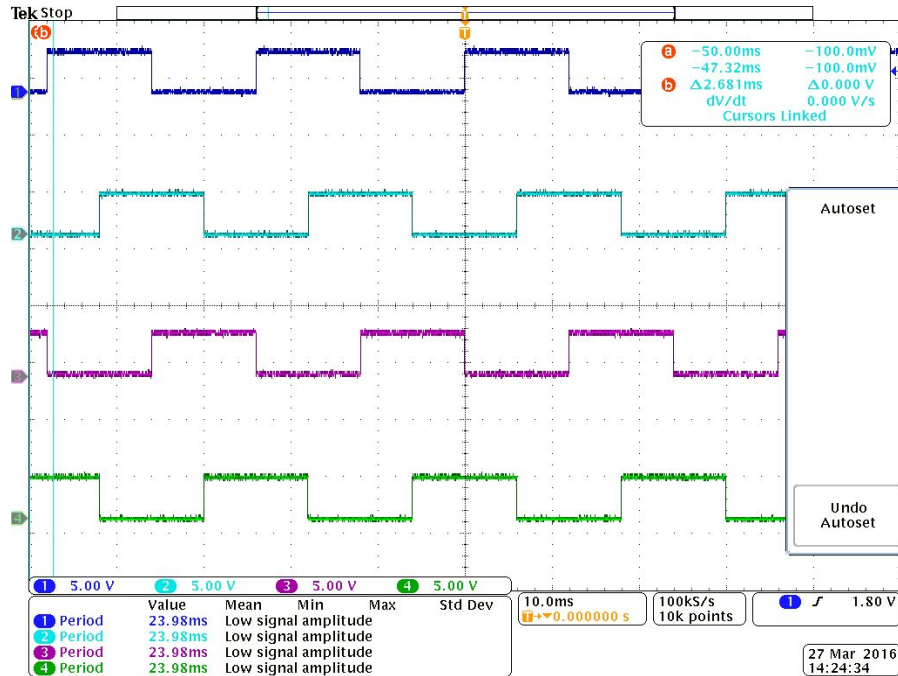


Figure 20: Full Step Control Signal Generated by Timer Interrupts

In this design, the timer interrupt has been used to control stepper motors. The stepper motor, which has been used in this design, has 200 steps/revolution with maximum speed 50 RPM. Therefore, the maximum speed should be 10000 steps/min and 167 steps/second. For timers, this design use Timer\_A0 and Timer\_A1 in MSP430G2553. The Timer\_A control register is shown in Figure 21.

15	14	13	12	11	10	9	8
<b>Unused</b>						<b>TASSELx</b>	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
<b>IDx</b>		<b>MCx</b>		<b>Unused</b>	<b>TACLR</b>	<b>TAIE</b>	<b>TAIFG</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
<b>Unused</b>	Bits 15-10	Unused					
<b>TASSELx</b>	Bits 9-8	Timer_A clock source select					
		00	TACLK				
		01	ACLK				
		10	SMCLK				
		11	INCLK (INCLK is device-specific and is often assigned to the inverted TBCLK) (see the device-specific data sheet)				
<b>IDx</b>	Bits 7-6	Input divider. These bits select the divider for the input clock.					
		00	/1				
		01	/2				
		10	/4				
		11	/8				
<b>MCx</b>	Bits 5-4	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power.					
		00	Stop mode: the timer is halted.				
		01	Up mode: the timer counts up to TACCR0.				
		10	Continuous mode: the timer counts up to 0FFFFh.				
		11	Up/down mode: the timer counts up to TACCR0 then down to 0000h.				
<b>Unused</b>	Bit 3	Unused					
<b>TACLR</b>	Bit 2	Timer_A clear. Setting this bit resets TAR, the clock divider, and the count direction. The TACLR bit is automatically reset and is always read as zero.					
<b>TAIE</b>	Bit 1	Timer_A interrupt enable. This bit enables the TAIFG interrupt request.					
		0	Interrupt disabled				
		1	Interrupt enabled				
<b>TAIFG</b>	Bit 0	Timer_A interrupt flag					
		0	No interrupt pending				
		1	Interrupt pending				

Figure 21: Timer\_A Control Register

The SMCLK clock source and up mode has been used. The up mode triggers the interrupt when the timer counts to the TACCR0 value, and at the same time the TACCR0 CCIFG interrupt flag will be set. Therefore, the TACTL will be set to TASSEL\_2 + MC\_1 + TACLR, where TACLR is used to reset the TAR, and TACLR bit is automatically reset to zero after the TAR has been reset. To enable the timer interrupt, the TACCTL, capture/compare control register, should set the CCIE, capture/compare interrupt enable bit, to 1. The capture/compare control register is shown in Figure 22.



15	14	13	12	11	10	9	8
<b>CMx</b>		<b>CCISx</b>		<b>SCS</b>	<b>SCCI</b>	<b>Unused</b>	<b>CAP</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
<b>OUTMODx</b>			<b>CCIE</b>	<b>CCI</b>	<b>OUT</b>	<b>COV</b>	<b>CCIFG</b>
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)
<b>CMx</b>	Bit 15-14	Capture mode					
		00	No capture				
		01	Capture on rising edge				
		10	Capture on falling edge				
		11	Capture on both rising and falling edges				
<b>CCISx</b>	Bit 13-12	Capture/compare input select. These bits select the TACCRx input signal. See the device-specific data sheet for specific signal connections.					
		00	CCxA				
		01	CCxB				
		10	GND				
		11	V <sub>CC</sub>				
<b>SCS</b>	Bit 11	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.					
		0	Asynchronous capture				
		1	Synchronous capture				
<b>SCCI</b>	Bit 10	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit					
<b>Unused</b>	Bit 9	Unused. Read only. Always read as 0.					
<b>CAP</b>	Bit 8	Capture mode					
		0	Compare mode				
		1	Capture mode				
<b>OUTMODx</b>	Bits 7-5	Output mode. Modes 2, 3, 6, and 7 are not useful for TACCR0, because EQUx = EQU0.					
		000	OUT bit value				
		001	Set				
		010	Toggle/reset				
		011	Set/reset				
		100	Toggle				
		101	Reset				
		110	Toggle/set				
		111	Reset/set				
<b>CCIE</b>	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag.					
		0	Interrupt disabled				
		1	Interrupt enabled				
<b>CCI</b>	Bit 3	Capture/compare input. The selected input signal can be read by this bit.					
<b>OUT</b>	Bit 2	Output. For output mode 0, this bit directly controls the state of the output.					
		0	Output low				
		1	Output high				
<b>COV</b>	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software.					
		0	No capture overflow occurred				
		1	Capture overflow occurred				
<b>CCIFG</b>	Bit 0	Capture/compare interrupt flag					
		0	No interrupt pending				
		1	Interrupt pending				

Figure 22: TACCTLx, Capture/Compare Control Register

The final stepper motor controlling API, `stepper_motor_driver.c` and `stepper_motor_driver.h`, are shown in Appendix B.

### 3.3.3 IR Sensor Control

The most important part of the firmware of the IR sensor is the ADC, and there are 8 channels of 10-bit analog-to-digital conversion. The INCHx is used to select ADC channels, and INCHx covers bit 12 to bit 15 in ADC10 Control Register 1, which is shown in Figure 23. Therefore, code “ADC10CTL1 = (pin << 12);”, where pin is the channel number of the ADC, has been used to select the ADC input channel. Another bit needed in this project is ADC10BUSY in ADC10 Control Register 1, which indicates an active sample or conversion operation. The ADC10BUSY bit has been used to check if conversion is completed.

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-0
Can be modified only when ENC = 0							

<b>INCHx</b>	Bits 15-12	Input channel select. These bits select the channel for a single-conversion or the highest channel for a sequence of conversions.
	0000	A0
	0001	A1
	0010	A2
	0011	A3
	0100	A4
	0101	A5
	0110	A6
	0111	A7
	1000	V <sub>REF+</sub>
	1001	V <sub>REF</sub> /V <sub>REF-</sub>
	1010	Temperature sensor
	1011	(V <sub>CC</sub> - V <sub>SS</sub> ) / 2
	1100	(V <sub>CC</sub> - V <sub>SS</sub> ) / 2, A12 on MSP430x22xx devices
	1101	(V <sub>CC</sub> - V <sub>SS</sub> ) / 2, A13 on MSP430x22xx devices
	1110	(V <sub>CC</sub> - V <sub>SS</sub> ) / 2, A14 on MSP430x22xx devices
	1111	(V <sub>CC</sub> - V <sub>SS</sub> ) / 2, A15 on MSP430x22xx devices
<b>ADC10BUSY</b>	Bit 0	ADC10 busy. This bit indicates an active sample or conversion operation
	0	No operation is active.
	1	A sequence, sample, or conversion is active.

Figure 23: Part of ADC10 Control Register 1

The Analog Input Enable Control Register 0 is used to enable the corresponding pin for analog input. The code looks like “ADC10AE0 = (1 << pin);” and the register is shown in Figure 24.

All of the data can be retrieved from 10 bit conversion results which is inside of the Conversion-Memory Register, which is shown in Figure 25.

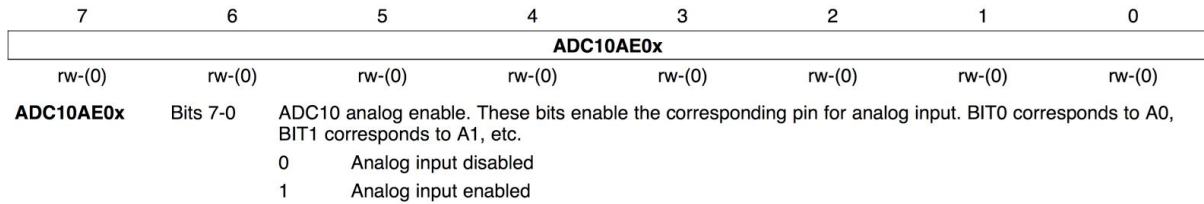


Figure 24: ADC10AE0, Analog (Input) Enable Control Register 0

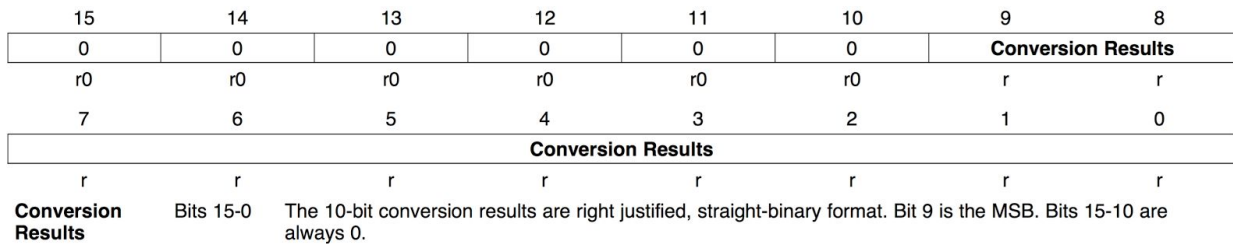


Figure 25: ADC10MEM, Conversion-Memory Register, Binary Format

There is another control register called ADC10 Control Register 0, which is used to select the reference voltage, timers and other ADC settings, and shown in figure 26. In this project, this register has been set to use the reference voltages as Vcc and Vss, 16 clock sample-and hold time. The ADC10ON is used to enable ADC, and the ENC and ADC10SC are used to enable and start conversion. The ADC code, adc.c and adc.h, are shown in Appendix B. The IR sensor API also includes the ir\_sensor\_driver code, whose c file and h file are shown in Appendix B, are used to get the data, and make sure the message, which will be sent out through the SPI Bus, is ready.

15	14	13	12	11	10	9	8
SREFx			ADC10SHTx		ADC10SR	REFOUT	REFBURST
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ENC = 0

<b>SREFx</b>	Bits 15-13	Select reference 000 $V_{R+} = V_{CC}$ and $V_{R-} = V_{SS}$ 001 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{SS}$ 010 $V_{R+} = V_{\theta REF+}$ and $V_{R-} = V_{SS}$ 011 $V_{R+} = \text{Buffered } V_{\theta REF+}$ and $V_{R-} = V_{SS}$ 100 $V_{R+} = V_{CC}$ and $V_{R-} = V_{REF} / V_{\theta REF-}$ 101 $V_{R+} = V_{REF+}$ and $V_{R-} = V_{REF} / V_{\theta REF-}$ 110 $V_{R+} = V_{\theta REF+}$ and $V_{R-} = V_{REF} / V_{\theta REF-}$ 111 $V_{R+} = \text{Buffered } V_{\theta REF+}$ and $V_{R-} = V_{REF} / V_{\theta REF-}$
<b>ADC10SHTx</b>	Bits 12-11	ADC10 sample-and-hold time 00 $4 \times \text{ADC10CLKs}$ 01 $8 \times \text{ADC10CLKs}$ 10 $16 \times \text{ADC10CLKs}$ 11 $64 \times \text{ADC10CLKs}$
<b>ADC10SR</b>	Bit 10	ADC10 sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC10SR reduces the current consumption of the reference buffer. 0 Reference buffer supports up to ~200 ksps 1 Reference buffer supports up to ~50 ksps
<b>REFOUT</b>	Bit 9	Reference output 0 Reference output off 1 Reference output on
<b>REFBURST</b>	Bit 8	Reference burst. 0 Reference buffer on continuously 1 Reference buffer on only during sample-and-conversion
<b>MSC</b>	Bit 7	Multiple sample and conversion. Valid only for sequence or repeated modes. 0 The sampling requires a rising edge of the SHI signal to trigger each sample-and-conversion. 1 The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed
<b>REF2_5V</b>	Bit 6	Reference-generator voltage. REFON must also be set. 0 1.5 V 1 2.5 V
<b>REFON</b>	Bit 5	Reference generator on 0 Reference off 1 Reference on
<b>ADC10ON</b>	Bit 4	ADC10 on 0 ADC10 off 1 ADC10 on
<b>ADC10IE</b>	Bit 3	ADC10 interrupt enable 0 Interrupt disabled 1 Interrupt enabled
<b>ADC10IFG</b>	Bit 2	ADC10 interrupt flag. This bit is set if ADC10MEM is loaded with a conversion result. It is automatically reset when the interrupt request is accepted, or it may be reset by software. When using the DTC this flag is set when a block of transfers is completed. 0 No interrupt pending 1 Interrupt pending
<b>ENC</b>	Bit 1	Enable conversion 0 ADC10 disabled 1 ADC10 enabled
<b>ADC10SC</b>	Bit 0	Start conversion. Software-controlled sample-and-conversion start. ADC10SC and ENC may be set together with one instruction. ADC10SC is reset automatically. 0 No sample-and-conversion start 1 Start sample-and-conversion

Figure 26: ADC10CTL0, ADC10 Control Register 0

### 3.4 Central Controller Setup

Central controller for this robot is built with a Raspberry Pi 2, which can be connected by computers, tablets, or smart phones through a Wi-Fi connection. The Raspberry Pi acts as the “hostap” (host access point). To set up the Raspberry Pi as hostap, software hostapd and isc-dhcp-server should be installed into the Raspberry Pi by using the following command.

```
Sudo apt-get install hostapd isc-dhcp-server
```

Host Access Point Daemon (Hostapd) is a user space software access point capable of turning normal network interface cards into access points and authentication servers [8]. The Dynamic Host Configuration Protocol (DHCP) is a network service that enables host computers to be automatically assigned settings from a server as opposed to manually configuring each network host. After installing the softwares, the setup work needs to be done. To set up the DHCP server, the dhcpd.conf file, which can be found under /etc/dhcp/dhcpd.conf, needs to be edited. In the dhcpd.conf file, find and comment out the following lines:

```
option domain-name "example.org";  
option domain-name-servers ns1.example.org, ns2.example.org;
```

And remove the comment for “authoritative;” . Then, add the following lines at the bottom of config file.

```
subnet 192.168.42.0 netmask 255.255.255.0 {  
    range 192.168.42.10 192.168.42.50;  
    option broadcast-address 192.168.42.255;  
    option routers 192.168.42.1;  
    default-lease-time 600;  
    max-lease-time 7200;  
    option domain-name "local";  
    option domain-name-servers 8.8.8.8, 8.8.4.4;  
}
```

After changing the dhcpd.conf file, saving it. Another configuration file called isc-dhcp-server, which can be found under /etc/default/isc-dhcp-server must be edited. In the isc-dhcp-server file, the line “INTERFACES=”” was updated to “INTERFACES="wlan0"”. Then the wlan0 must be setup for a static IP, which is necessary to be known to remote connect to the robot. The command “sudo ifdown wlan0” is run to activate wlan0. Next the configuration file interfaces, which are under /etc/network/interfaces, are edited. The wlan0 settings are updated to

```
allow-hotplug wlan0
iface wlan0 inet static
    address 192.168.42.1
    netmask 255.255.255.0
```

The command “sudo ifconfig wlan0 192.168.42.1” is then run to assign a static IP address to the wifi adapter. The address need not be 192.168.42.1, it can be any ip address. In this project, 192.168.42.1 has been used as the IP address. After setting up the DHCP server, the next thing to do is configure the access point. For this project, the access point will set up a password-protected network so only people with the password can connect. The first step to configure the access point is creating or editing a configuration file named hostapd.conf under /etc/hostapd/hostapd.conf. And then adding or updating the following lines

```
interface=wlan0
driver=rtl871xdrv
ssid=Pi_Robot_AP
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=RaspberryPiRobot
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

where the ssid, the network broadcast name, and wps\_passphrase, the password for this access point, can be change to anything user want. The driver for the wifi adapters needs to be determined, and it will be set by “driver=”. The value of the driver in hostapd.conf could be rtl871xdrv, which used for Adafruit wifi adapters, nl80211 or something else. The Raspberry Pi need to be told where to find this configuration file, the “#DAEMON\_CONF=” in file /etc/default/hostapd needs to be updated to “DAEMON\_CONF="/etc/hostapd/hostapd.conf””. After this, the Network Address Translation (NAT) must be set up, which will allow multiple clients to connect to the Wi-Fi and have all the data tunneled through the single Ethernet IP. “net.ipv4.ip\_forward=1” must be added to the configuration file /etc/sysctl.conf. This will start IP forwarding on boot up. Then the command “sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip\_forward”” needs to be run to activate it immediately. Next, the following commands need to be run to create the network translation between the ethernet port eth0 and the wifi port wlan0.

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

After rebooting the changes will take effect by running the command

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

At the end of /etc/network/interfaces configuration file, the line “up iptables-restore < /etc/iptables.ipv4.nat” must be added, and the following commands run to start it.

```
sudo service hostapd start
sudo service isc-dhcp-server start
```

[9]. At this point, any kind of devices can connect to the Raspberry Pi through Wi-Fi with AP, and control the robot by ssh into the robot through the ip address.

The next step is to set up the programming environment, beginning with the c-periphery library. The c-periphery is a set of C wrapper functions, which wrap, simplify, and consolidate the native Linux APIs to GPIO, SPI, I2C, MMIO, and Serial peripheral I/O interface access in the embedded Linux environments (including BeagleBone, Raspberry Pi, etc. platforms). To install the c-periphery library, the c-periphery files need to be download into the Raspberry Pi, and the current directory must be c-periphery directory. The “make” command is used to build c-periphery into a static library and “make tests” can be used to build c-periphery tests. The “CC” environment variable must be set with the cross-compiler by use the following command.

```
CC=arm-linux-gcc make clean all tests
```

Now the library has been set up on the Raspberry Pi, and it is ready to use. Then the header files just need to be included in src/, such as gpio.h, spi.h, i2c.h, mmio.h, serial.h, and the periphery.a static library linked in when compiling the C control code for the robot. For example, to compile myprog.c program, the command

```
gcc -I/path/to/periphery/src myprog.c /path/to/periphery/periphery.a -o myprog
```

should be used to generate the output file named myprog.[10] As central controller of this robot is a Raspberry Pi which is running Linux, the robot can be programmed in any programming languages which can be compile by Linux. There are another two Linux API wrappers for these interfaces for embedded Linux environments named python-periphery and lua-periphery, and these libraries allow this robot to be controlled by python and lua.



### 3.5 Firmware Design for Central Controller

After setting up the programming environments on the central controller, the robot is ready for firmware design. In this design, firmware for the IR sensor and the stepper motor controller has been done. The firmware for the IR sensor and the stepper motor controller is a set of functions which is used to communicate with modules. It sends the commands out and gets the feedback message to control each modules and gets user input to control the robot. All functions which are used to control the robot in the central controller are designed by following the flowchart shown in Figure 27. All of the functions the robot has now are attached in Appendix C. Users can call these functions in their own program to control the robot.

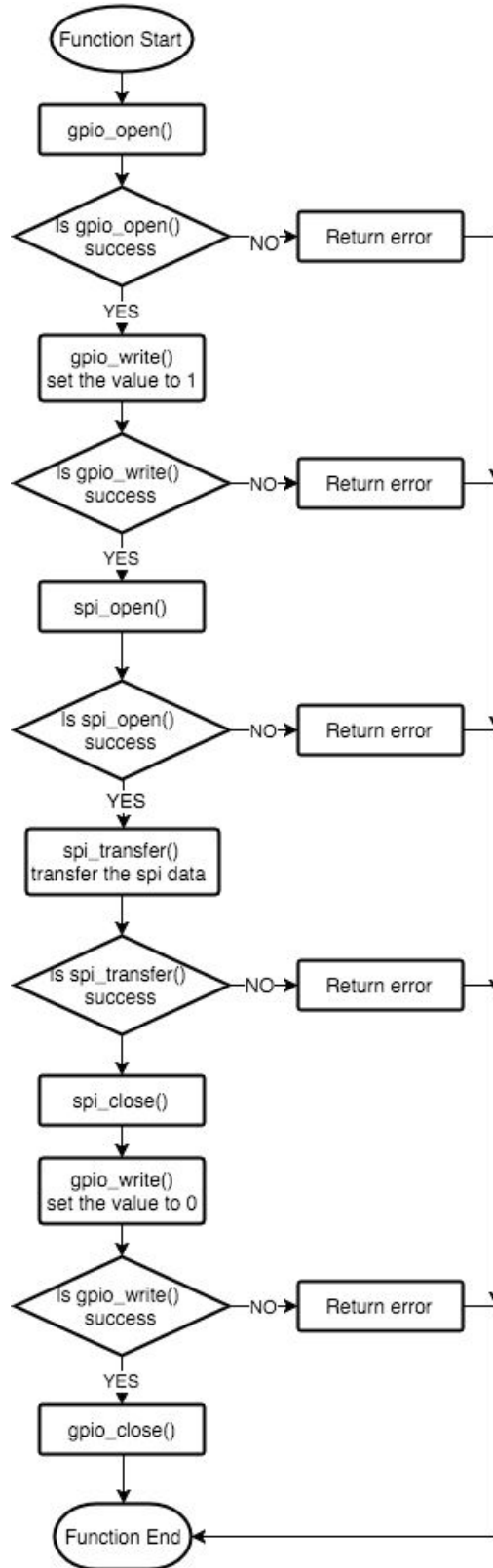


Figure 27: Flow Chart for a Basic Function Design

## 4. Demo Project

The demo projects are used to demonstrate that the robot has all of the features which other educational robot platforms have. There are three demo projects that illustrate that the robot platform works, they are basic control, obstacle avoidance, and wall following. Basic control is used to test simple features like motors go, get the IR sensor readings. Obstacle avoidance and wall following projects are used to demonstrate this robot platform can be used to design more advanced Artificial Intelligence (AI) Systems.

Basic control is a simple demo. The idea of this demo is to use a computer to control the robot platform by sending different commands. This demo uses the `stepper_motor_api` and `ir_sensor_api`. The `main.c` file of the basic control demo gets the control command through the command line input and calls the functions which have been given in `stepper_motor_api` and `ir_sensor_api`. The final output file, `robot`, would be generated by running the makefile. The robot platform can be controlled by running the code in the form “`./robot <control command> <other commands (option)>`”. For example, when instructing the robot to go forward with speed 50 rpm, the command “`./robot go 50 1 50 1`” is used, and the command “`./robot irget`” is used to get the IR sensor readings. All of the control commands are shown in Table 2. The makefile and `main.c` file are included in Appendix D.

**Table 2: Control Command for Basic Control Demo**

Commands	Description
<code>./robot go &lt;speed right&gt; &lt;direction right&gt; &lt;speed left&gt; &lt;direction left&gt;</code>	This command is used to run the robot at different speeds in rpm and different directions.
<code>./robot go</code>	This command is used to run the robot in default settings. It is the same as <code>./robot go 50 0 50 0</code> .

<code>./robot stop</code>	This command is used to stop the robot.
<code>./robot get</code>	This command is used to get the total steps travelled when robot stopped.
<code>./robot cget</code>	This command is used to get the total steps while the robot is running.
<code>./robot irget</code>	This command is used to get all four IR sensors values
<code>./robot irget &lt;sensor name&gt;</code>	This command is used to get the value of one of the four IR sensor

The demo of obstacle avoidance includes three small demos, stop before obstacle, shy avoid, and obstacle avoid. These three small demos are also three steps for the obstacle avoidance demo project. The first step shows that robot can get the data from IR sensor and it can make a response with the feedback from the IR sensor. The stop before obstacle demo shows robot can go forward and continue getting feedback from IR sensors. When the robot senses that there is an obstacle in front of it, it stops. This step shows the basic feature that the robot can respond and get feedback with interrupts, the next step is used to show that robot can execute tasks recursively. Shy avoid is a project to make the robot like a shy kid, when an obstacle is too close to the robot, it will run away. This code runs continuously and is example of subsumption architecture where the priority is a halt behavior that can be subsumed by a runaway behavior. The code has the robot check both the front and back IR sensors. If an obstacle is too close to the robot, the robot runs away. The obstacle avoid project requires the robot to do something to make it avoid the obstacle not only detect the obstacle. In this demo, the robot keeps going forward and making right turns when it senses an obstacle to avoid the obstacle. The makefile of this demo project generates three output files, they are StopBeforeObstacle, ShyAvoid, and

ObstacleAvoid. The demos are easy to run by using “./” to run the output files in sudo mode. All code is shown in Appendix D.

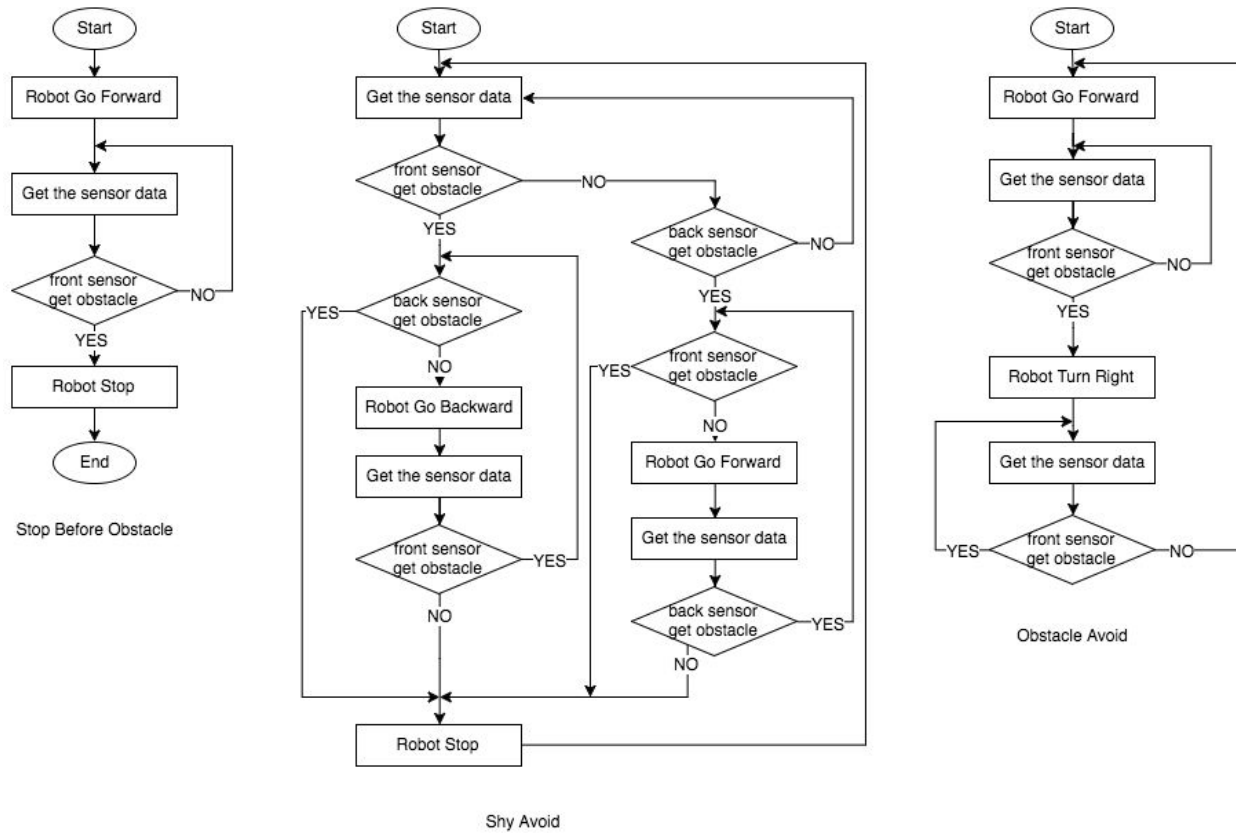


Figure 28: Flow Chart for Obstacle Avoid Project

The third demo project is wall following, it required the robot to follow walls and use P, PI, or PID feedback control to determine how to respond to inner and outer corners in the wall. This project has three part that are required to function, following the wall, turning on the inner corner, and turning on the outer corner. For following the wall, it used a proportional controller to set the robot’s distance to the wall. When the robot detects the corner, it distinguishes what type this corner is. If the front sensor senses the wall, the corner should be an inner corner, and if the sensor on the same side of the wall lose the wall, it should be a outer corner. When robot is in the inner corner, it makes a turn to the opposite wall, and keeps following the new wall which

was detected by the front sensor before. When robot is in the outer corner, the robot should run in a small circle and make the front sensor touch the new wall, and then turn to make the robot follow the wall again. The makefile generates an output file named WallFollowing. All of the project programs are shown in Appendix D.

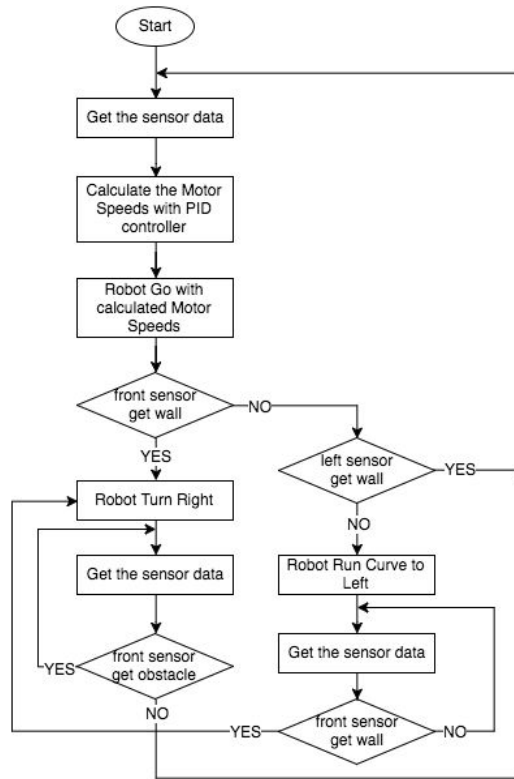


Figure 29: Flow Chart for Wall Following

All of the demos prove that this mobile robot platform works well and has all of the features which are required to design an AI systems. The videos of these demos are uploaded to Youtube and listed in Appendix E. This robotics platform should also be able to do the other AI behaviors such as line following, homing, docking, heat and light sensing, localization, path planning, and mapping by using current design. For these AI behaviors, it maybe useful to get the feedback of counts of stepper motor steps, which should be helpful to calculate the distance the robot goes.

## 5. Comparison

This mobile robot platform is designed for education, and it's based on all the features which ECE 425 Mobile Robotics class needed. The demo projects using this mobile robot platform proved that this mobile robot platform can be used in ECE 425 class for teaching students programming and control of a mobile robot. Comparing this mobile robot platform, Arduino Robot, which ECE 425 currently uses, this mobile robot platform is more powerful.

For hardware, due to the mobile robot platform modular design, the user can add any kind of hardware, such as GPIO, ADC, DAC, SPI, I2C, UART, by designing their own custom modules. The current design uses SPI bus, which requires an extra pin to select slave module, this robot platform can have 20 extra custom modules, which give very large capacity for users to add their own features into this robot. However, Arduino Robot is limited on the hardware side. The Arduino Robot only has 8 extra digital I/O/Analog input channels, 7 PWM channels, and some I2C ports [11]. However, the Arduino Robot has a nice full color LCD screen, speaker, pushbuttons, IR line follower sensors, compass, and motors, but the mobile robot platform designed in this thesis only has two motors and four IR sensors [11]. For motors, this mobile robot platform has two stepper motors, which can control the motions very accurately, but Arduino Robot only has two DC motor without encoders, which means all of the motions only can be controlled by timers or sensors [11]. Overall, the mobile robot platform designed in this thesis is much more powerful and gives more opportunity for users to design their own robot projects, but it has fewer fancy features, all of these features need users to design as a custom module and add it into the robot by themselves.

For programming the robot, this robot platform is unlike Arduino Robot in that it does not have an IDE, and all the programs are done by using text editors, such as vim. And it need the user to compile the program by themselves, such as writing makefiles or using the command lines to compile the program. If it has a IDE like Arduino Robot, the compile and program will becomes easier by just click a button. Currently, due to it does not have the IDE yet, and the Arduino IDE is very simple, the robot platform designed in this thesis is harder to program than Arduino Robot. However, Arduino Robot can be programed only in Arduino C code, but the robot platform designed in this thesis has the feature to program in different programming languages. Therefore, this robot platform is better in that it can use more than one programming language, and be programmed by different devices like smart phones. However, the Arduino Robot is better in that it has a nice IDE which makes user easier to program and debug the robot.

Designing a custom module require users to study embedded firmware, to learn how to program a microcontroller by using C or other low level programming languages, it increases the depth of knowledge level when users design their own projects by using the robot platform designed in this thesis. Therefore, it also gives more opportunity to learn how to program an embedded system. Compared with Arduino Robot, the Arduino Robot does not require user to do a lot of low level firmware program. Arduino is easy to use, it already contains a lot of useful functions. Users just need to call these functions and they do not need to know how these function works. However, in the real world, a good engineer in area of embedded system needs to know more about the lower level programming, which user cannot learn from using Arduino. Therefore, the robot platform designed in this thesis makes users learn more skills.



Overall, compared to the Arduino Robot, this modular educational mobile robot platform is much more powerful, which means it has more pins, more features, and can be programmed in different languages and different devices. However, it still has some shortages, such as the IDE, if it has a nice IDE, it would make design works much easier.

## 6. Conclusion and Future Work

Recall that the purpose of this thesis was to design a modular educational robotic platform which could solve the limitation problems which current educational robot platforms have, such as limited number of I/O pins, only use low level programming languages, and only can be programmed by PCs. The goal of this modular educational robotic platform was to solve the limited pins, single programming language, and single programming device problems. The feedback given by a student volunteer who used this modular educational robotic platform was said, “this robot platform is simple and powerful”. The Linux System makes the robot easily controlled by different devices, and it can be programed in different programming languages. The modular design makes it easy to add more hardware, and the plug and play design is interesting and easy to use. Also, three demos show that this robot platform has all the features which are needed for education. Overall, this modular educational robotic platform solves the problems by providing more pins, multiple programming language, and multiple programming devices.

Future work includes designing an Integrated Development Environment (IDE), which can support different requirements for different development levels. The elementary version of the IDE is preferred to be a block programming tool which likes LEGO programming IDE. Users just need to drag and drop different blocks into the programming area, and draw lines to link different blocks. Then the robot will run following the sequence of blocks. This IDE will have another version for more advanced robotics designers and programmers, it should look similar to the Arduino IDE. When the user opens a new main file, there are two default functions named

init() and loop(). The init() function will run at the beginning of the code, and the loop() function will run inside a main loop. The last version of this IDE is used for users at a professional level without Linux experience. This version is just like a normal C programming IDE like Eclipse, users can fully with code and use buttons to build, compile and execute the code. This IDE will make it easier for users to program and control the robot at a more advanced level.

Other future work is to create a battery monitoring system for the robot. In the current design, there is no battery monitoring system inside the robot, which means the user does not know if the battery level is too low until it turns the robot power off. Fully discharging will reduce the battery life for Li-ion battery, so a battery monitoring system is needed to track the battery level to protect battery life. Also, SPI communication will become unstable when the battery level is too low, so it good to let the user know about the battery level to help troubleshooting. Therefore, a battery monitoring system makes the robot more user friendly and protects the robot. Also a AI algorithms to sensing the light will be added to the future, which will require adding a lighting sensor module, to help understand how to using this platform.

## LIST OF REFERENCES

- [1] Shvartsman, Andrey, Maurice Tedder, and Chan-Jin Chung. "A Modular Mobile Robotic Platform As An Educational Tool In Computer Science And Engineering." Web.
- [2] "Robobutler." *Raspberry Pi*. Web. <<https://www.raspberrypi.org/learning/robo-butler/>>.
- [3] Piperidis, S., L. Doitsidis, C. Anastasopoulos, and N. C. Tsourveloudis. "A Low Cost Modular Robot Vehicle Design for Research and Education." *2007 Mediterranean Conference on Control & Automation* (2007).
- [4] Tur, J.m. Mirats, and C.f. Pfeiffer. "Mobile Robot Design in Education." *IEEE Robotics & Automation Magazine IEEE Robot. Automat. Mag.* 13.1 (2006): 69-75.
- [5] Berry, Carlotta A. *Mobile Robotics for Multidisciplinary Study*. San Rafael, Calif.?: Morgan & Claypool, 2012. Print.
- [6] Gibson, Steve. "Tech Talk." *InfoWorld* 1 Dec. 1986: 70. Print.
- [7] "Stepper Motor Basics." *Solarbotics.net*. Web.  
<<http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf>>.
- [8] Malinen, Jouni. "Hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator." *Hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator*. Web. 12 Jan. 2013. <<http://w1.fi/hostapd/>>.
- [9] Ada, Lady. "Setting up a Raspberry Pi as a WiFi Access Point." *Adafruit*. Web. 20 Nov. 2015.
- [10] Sergeev, Vanya. "Vsergeev/c-periphery." *GitHub*. Web. 8 Aug. 2015.  
<<https://github.com/vsergeev/c-periphery>>.
- [11] "Arduino - Robot." *Arduino - Robot*. Web. <<https://www.arduino.cc/en/Main/Robot>>.



## APPENDICES

### APPENDIX A: PROJECT PARTS LIST

	Part Number	Description	Price	Quantity	Total
Capacitors	CL21B102KBANNNC	1nF 0805 Capacitor	\$0.10	3	\$0.30
	CL21F104ZBCNNNC	0.1uF 0805 Capacitor	\$0.10	5	\$0.50
	CL21F105ZOCNNNC	1uF 0805 Capacitor	\$0.10	6	\$0.60
	CL21F106ZPFNNNE	10uF 0805 Capacitor	\$0.16	3	\$0.48
	UWX0G221MCL1GB	220uF Electric Capacitor	\$0.33	2	\$0.66
Resistors	RC2012F331CS	330 Ohm 0805 Resistor	\$0.10	2	\$0.20
	RC0805JR-0747KL	47K Ohm 0805 Resistor	\$0.10	3	\$0.30
	RC0805JR-0710KL	10K Ohm 0805 Resistor	\$0.10	8	\$0.80
LEDs	LG R971-KN-1	LED GREEN DIFFUSED 0805 SMD	\$0.26	2	\$0.52
ICs	MSP430G2553IPW28R	MSP430G2553 28Pin MCU TSSOP-28	\$2.73	2	\$5.46
	MSP430G2553IPW20R	MSP430G2553 20Pin MCU TSSOP-20	\$2.59	1	\$2.59
	L293DD	H-Bridge Motor Controller	\$3.69	2	\$7.38
	LM1117IMPX-3.3/NOPB	Linear Voltage Regulator	\$1.06	1	\$1.06

	V7805-2000R	CONVERT DC/DC REG 5V 2000MA R/A	\$10.84	1	\$10.84
Connectors	640454-4	CONN HEADER VERT 4POS .100 TIN	\$0.18	2	\$0.36
	0022013047	CONN HOUS 4POS .100 W/RAMP/RIB	\$0.20	2	\$0.40
	640454-3	CONN HEADER VERT 3POS .100 TIN	\$0.16	4	\$0.64
	0022013037	CONN HOUS 3POS .100 W/RAMP/RIB	\$0.19	4	\$0.76
	640456-7	CONN HEADER VERT 7POS .100 TIN	\$0.27	1	\$0.27
	0022053071	CONN HEADER VERT 7POS .100 TIN (Right Angle)	\$0.79	1	\$0.79
	0022013077	CONN HOUS 7POS .100	\$0.36	2	\$0.72
	0008500114	CONN TERM FEMALE 22-30AWG TIN	\$0.19	34	\$6.46
	Adafruit/P2223	GPIO Stacking Header For Raspberry Pi	\$2.50	1	\$2.50
	1935161	TERM BLOCK PCB 2POS 5.0MM GREEN	\$0.37	1	\$0.37
		.100" pitch Connector Headers	\$0.50	1	\$0.50
		USB Connector	\$0.10	1	\$0.10

Pushbutton		SWITCH TACTILE			
	FSMSM	SPST-NO 0.05A 24V	\$0.22	3	\$0.66
Motors	Adafruit/324	Stepper Motor	\$14.00	2	\$28.00
Others		Raspberrry Pi 2	\$40.00	1	\$40.00
		Battery	\$23.92	1	\$23.92
				Total:	\$138.14



## APPENDIX B: MSP430G2553 FIRMWARE CODE

spi\_slave.h

```
#ifndef _SPI_H_
#define _SPI_H_

char txbuf[10];

void spi_init(void);
void spi_putc(unsigned char c);
void return_message(char *buf);

#endif
```

spi\_slave.c

```
#include <msp430g2553.h>
#include "spi_slave.h"
#include "commands.h"

char cmdbuf[10];
char cmd_index=0;

void spi_init(void) {
    P1SEL |= BIT4 | BIT5 | BIT6 | BIT7;
    P1SEL2 |= BIT4 | BIT5 | BIT6 | BIT7;
    UCB0CTL1 = UCSWRST;
    UCB0CTL0 |= UCMSB + UCSYNC + UCCKPH + UCMODE_1;
    UCB0CTL1 &= ~UCSWRST;
    IE2 |= UCB0RXIE;
}

void spi_putc(unsigned char c) {
    while (!(IFG2 & UCB0TXIFG));
    UCB0TXBUF = c;
}

void return_message(char *buf) {
    int i = 0;
    while (*buf) {
        txbuf[i] = *buf++;
        i++;
    }
    spi_putc(txbuf[0]);
}

#pragma vector = USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {
    char rx_char;

    while (IFG2 & UCB0RXIFG) {
        rx_char = UCB0RXBUF;
        if (rx_char == '\n') {
            commands(cmdbuf);
            cmd_index = 0;
        } else {
            cmdbuf[cmd_index] = rx_char;
            cmd_index++;
            spi_putc(txbuf[cmd_index]);
        }
    }
}
```

}  
}  
}

stepper\_motor\_driver.h

```
#ifndef _STEPPER_MOTOR_DRIVER_H
#define _STEPPER_MOTOR_DRIVER_H

#include <stdint.h>

/* Define step motor pin */
#define STEPPERMOTOR_PORTOUT P3OUT
#define STEPPERMOTOR_PORTDIR P3DIR
#define STEPPERMOTOR_PORTREN P3REN
//define the right motor pin
#define STEPPERMOTOR_MR1A BIT0
#define STEPPERMOTOR_MR1B BIT1
#define STEPPERMOTOR_MR2A BIT2
#define STEPPERMOTOR_MR2B BIT3
//define the left motor pin
#define STEPPERMOTOR_ML1A BIT4
#define STEPPERMOTOR_ML1B BIT5
#define STEPPERMOTOR_ML2A BIT6
#define STEPPERMOTOR_ML2B BIT7

//define some constant values
#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1
#define BOTH_MOTOR 2
#define DIRECTION_FORWARD 0
#define DIRECTION_BACKWARD 1
#define DIRECTION_CLOCWISE 0
#define DIRECTION_COUNTERCLOCKWISE 1

void stepper_motor_init(void);
void robot_go(uint8_t speed_R, uint8_t direction_R, uint8_t speed_L, uint8_t
direction_L);
void robot_stop(void);
void robot_get_cycle_count(void);

#endif
```

## Stepper\_motor\_driver.c

```
#include <msp430.h>
#include <stdlib.h>
#include "stepper_motor_driver.h"
#include "spi_slave.h"

#define WHEEL_DIFFERENCE 200 //wheel differences in mm
#define WHEEL_DIAMETER 85 //wheel diameter in mm
#define WHEEL_PERIMETER 267 //wheel perimeter in mm
#define PI 3

volatile uint8_t DIRECTION_R = 0;
volatile uint8_t DIRECTION_L = 0;
volatile uint8_t MOTOR_SPEED_R = 0;
volatile uint8_t MOTOR_SPEED_L = 0;
volatile uint32_t CYCLES_COUNT_R = 0;
volatile uint32_t CYCLES_COUNT_L = 0;

/***** stepper_motor_init() *****/
Enable the motor pin settings and Timer settings
*****/
void stepper_motor_init(void) {
    //init the io pins for all stepper motors
    STEPPERMOTOR_PORTOUT &= (~STEPPERMOTOR_MR1A) &
(~STEPPERMOTOR_MR1B)
        & (~STEPPERMOTOR_MR2A) & (~STEPPERMOTOR_MR2B)
& (~STEPPERMOTOR_ML1A)
        & (~STEPPERMOTOR_ML1B) & (~STEPPERMOTOR_ML2A)
        & (~STEPPERMOTOR_ML2B);
    STEPPERMOTOR_PORTDIR |= STEPPERMOTOR_MR1A |
STEPPERMOTOR_MR1B
        | STEPPERMOTOR_MR2A | STEPPERMOTOR_MR2B |
STEPPERMOTOR_ML1A
        | STEPPERMOTOR_ML1B | STEPPERMOTOR_ML2A |
STEPPERMOTOR_ML2B;
    STEPPERMOTOR_PORTREN &= (~STEPPERMOTOR_MR1A) &
(~STEPPERMOTOR_MR1B)
        & (~STEPPERMOTOR_MR2A) & (~STEPPERMOTOR_MR2B)
& (~STEPPERMOTOR_ML1A)
        & (~STEPPERMOTOR_ML1B) & (~STEPPERMOTOR_ML2A)
        & (~STEPPERMOTOR_ML2B);

    //TIMER A CONFIG
    TA0CTL = TASSEL_2 | MC_0 | TACL;
}
```

```

    TA1CTL = TASSEL_2 | MC_0 | TACLR;
    TA0CCTL0 &= ~CCIE;
    TA1CCTL0 &= ~CCIE;
}

/***** stepper_motor_direction_set() *****/
Set the motor run direction
motor_sel: motor selection 0:right 1:left 2:all
direction: motor direction 1:front 0:back
*****/
void stepper_motor_direction_set(uint8_t motor_sel, uint8_t direction) {
    switch (motor_sel) {
        case RIGHT_MOTOR:
            DIRECTION_R = direction;
            break;
        case LEFT_MOTOR:
            DIRECTION_L = direction;
            break;
        default:
            DIRECTION_R = direction;
            DIRECTION_L = direction;
            break;
    }
}

/***** set_speed() *****/
set the motor speed
motor_sel: motor selection 0:right 1:left 2:all
speed: motor speed 5~50 rpm
*****/
void set_speed(uint8_t motor_sel, uint8_t speed) {
    switch (motor_sel) {
        case RIGHT_MOTOR:
            MOTOR_SPEED_R = speed;
            break;
        case LEFT_MOTOR:
            MOTOR_SPEED_L = speed;
            break;
        default:
            MOTOR_SPEED_R = speed;
            MOTOR_SPEED_L = speed;
            break;
    }
}

```

```

/***** calculate_speed_cycles() *****/
calculate and return the timer cycles by motor
speed in rpm
*****/
uint16_t calculate_speed_cycles(uint8_t speed) {
    uint16_t steps = 0;
    uint16_t delay_cycles = 0;
    uint8_t i = 0;
    for (i = 0; i < speed; i++) {
        steps += 200;
    }
    delay_cycles = (uint16_t)(60000000 / steps);
    return delay_cycles;
}

/***** timer_enable() *****/
enable the timer interrupt
*****/
void timer_enable(void) {
    TA0CTL0 |= CCIE;
    TA0CCR0 = calculate_speed_cycles(MOTOR_SPEED_R);

    TA1CTL0 |= CCIE;
    TA1CCR0 = calculate_speed_cycles(MOTOR_SPEED_L);

    TA0CTL = TASSEL_2 | MC_1 | TACLK;
    TA1CTL = TASSEL_2 | MC_1 | TACLK;
}

/***** timer_disable() *****/
disable the timer interrupt
*****/
void timer_disable(void) {
    TA0CTL0 &= ~CCIE;
    TA1CTL0 &= ~CCIE;

    TA0CTL = MC_0;
    TA1CTL = MC_0;
}

/***** robot_go() *****/
the robot will go by different speed
speed_R, speed_L: motor speed 5~50 rpm
direction_R, direction_L: 0:front 1:back
*****/

```

```

void robot_go(uint8_t speed_R, uint8_t direction_R, uint8_t speed_L, uint8_t
direction_L) {
    CYCLES_COUNT_R = 0;
    CYCLES_COUNT_L = 0;
    stepper_motor_direction_set(RIGHT_MOTOR, direction_R);
    stepper_motor_direction_set(LEFT_MOTOR, direction_L);
    set_speed(RIGHT_MOTOR, speed_R);
    set_speed(LEFT_MOTOR, speed_L);
    timer_enable();
    return_message("M:RUNNING.");
}

/***** robot_stop() *****/
the robot will stop
*****/
void robot_stop(void) {
    timer_disable();
    set_speed(RIGHT_MOTOR, 0);
    set_speed(LEFT_MOTOR, 0);
    unsigned char *rightCount = (unsigned char*)&CYCLES_COUNT_R;
    unsigned char *leftCount = (unsigned char*)&CYCLES_COUNT_L;
    char buf[10] = {'M', ':', rightCount[0]+1, rightCount[1]+1, rightCount[2]+1,
rightCount[3]+1, leftCount[0]+1, leftCount[1]+1, leftCount[2]+1, leftCount[3]+1};
    return_message((char *)buf);
}

/***** robot_get_cycle_count() *****/
the robot will return the cycle count
*****/
void robot_get_cycle_count(void) {
    unsigned char *rightCount = (unsigned char*)&CYCLES_COUNT_R;
    unsigned char *leftCount = (unsigned char*)&CYCLES_COUNT_L;
    char buf[10] = {'M', ':', rightCount[0]+1, rightCount[1]+1, rightCount[2]+1,
rightCount[3]+1, leftCount[0]+1, leftCount[1]+1, leftCount[2]+1, leftCount[3]+1};
    return_message((char *)buf);
}

//TIMER A INTERRUPT A0 for right, A1 for left
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0(void) {
    if (DIRECTION_R == 0) {
        switch(CYCLES_COUNT_R % 4) {
            case 0:
                STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2B) & (~STEPPERMOTOR_MR1A);

```



```

        STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2A | STEPPERMOTOR_MR1B;
        break;
        case 1:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2B) & (~STEPPERMOTOR_MR1B);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2A | STEPPERMOTOR_MR1A;
            break;
        case 2:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2A) & (~STEPPERMOTOR_MR1B);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2B | STEPPERMOTOR_MR1A;
            break;
        case 3:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2A) & (~STEPPERMOTOR_MR1A);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2B | STEPPERMOTOR_MR1B;
            break;
        default:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2B) & (~STEPPERMOTOR_MR2A) &
(~STEPPERMOTOR_MR1B) & (~STEPPERMOTOR_MR1A);
            break;
    }
} else {
    switch(CYCLES_COUNT_R % 4) {
        case 0:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2A) & (~STEPPERMOTOR_MR1A);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2B | STEPPERMOTOR_MR1B;
            break;
        case 1:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2A) & (~STEPPERMOTOR_MR1B);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2B | STEPPERMOTOR_MR1A;
            break;
        case 2:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2B) & (~STEPPERMOTOR_MR1B);

```

```

        STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2A | STEPPERMOTOR_MR1A;
        break;
    case 3:
        STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2B) & (~STEPPERMOTOR_MR1A);
        STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_MR2A | STEPPERMOTOR_MR1B;
        break;
    default:
        STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_MR2B) & (~STEPPERMOTOR_MR2A) &
(~STEPPERMOTOR_MR1B) & (~STEPPERMOTOR_MR1A);
        break;
    }
}
CYCLES_COUNT_R++;
}

```

```

#pragma vector=TIMER1_A0_VECTOR
__interrupt void Timer_A1(void) {
    if (DIRECTION_L == 0) {
        switch(CYCLES_COUNT_L % 4) {
            case 0:
                STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2A) & (~STEPPERMOTOR_ML1A);
                STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2B | STEPPERMOTOR_ML1B;
                break;
            case 1:
                STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2A) & (~STEPPERMOTOR_ML1B);
                STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2B | STEPPERMOTOR_ML1A;
                break;
            case 2:
                STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2B) & (~STEPPERMOTOR_ML1B);
                STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2A | STEPPERMOTOR_ML1A;
                break;
            case 3:
                STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2B) & (~STEPPERMOTOR_ML1A);

```

```

        STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2A | STEPPERMOTOR_ML1B;
        break;
    default:
        STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2B) & (~STEPPERMOTOR_ML2A) &
(~STEPPERMOTOR_ML1B) & (~STEPPERMOTOR_ML1A);
        break;
    }
} else {
    switch(CYCLES_COUNT_L % 4) {
        case 0:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2B) & (~STEPPERMOTOR_ML1A);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2A | STEPPERMOTOR_ML1B;
            break;
        case 1:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2B) & (~STEPPERMOTOR_ML1B);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2A | STEPPERMOTOR_ML1A;
            break;
        case 2:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2A) & (~STEPPERMOTOR_ML1B);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2B | STEPPERMOTOR_ML1A;
            break;
        case 3:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2A) & (~STEPPERMOTOR_ML1A);
            STEPPERMOTOR_PORTOUT |=
STEPPERMOTOR_ML2B | STEPPERMOTOR_ML1B;
            break;
        default:
            STEPPERMOTOR_PORTOUT &=
(~STEPPERMOTOR_ML2B) & (~STEPPERMOTOR_ML2A) &
(~STEPPERMOTOR_ML1B) & (~STEPPERMOTOR_ML1A);
            break;
    }
}
CYCLES_COUNT_L++;
}

```

adc.h

```
#ifndef _ADC_H_
#define _ADC_H_

void init_adc(void);
uint16_t adc_convert(uint8_t pin);

#endif /* _ADC_H_ */
```

adc.c

```
#include <msp430.h>
#include <stdint.h>
#include "adc.h"

void init_adc(void) {
    uint16_t adc10ctl0 = 0x0000;
    adc10ctl0 |= SREF_0; /* VR+ = VCC & VR- = VSS */
    adc10ctl0 |= ADC10SHT_2; /* 16 clock sample-and-hold time */
    adc10ctl0 |= ADC10ON; /* enable ADC */
    ADC10CTL0 = adc10ctl0;
}

/* 10-bit ADC will return max of 0x3FF */
uint16_t adc_convert(uint8_t pin) {

    ADC10CTL1 = (pin << 12);
    ADC10AE0 = (1 << pin); /* enable analog input */

    ADC10CTL0 |= (ENC | ADC10SC); /* enable & start conversion */
    while(ADC10CTL1 & ADC10BUSY); /* wait for conversion to complete */
    ADC10CTL0 &= ~ENC; /* disable conversion */

    return(ADC10MEM); /* converted value */
}
```

ir\_sensor\_driver.h

```
#ifndef _IR_SENSOR_DRIVER_H_
#define _IR_SENSOR_DRIVER_H_

#define IR_SENSOR_F 0
#define IR_SENSOR_B 1
#define IR_SENSOR_L 2
#define IR_SENSOR_R 3

void ir_sensor_init(void);
uint16_t get_IR_sensor_data(uint8_t IR_number);
void prepare_IR_data(void);
void get_IR_data(void);

#endif
```

ir\_sensor\_driver.c

```
#include <stdint.h>
#include "ir_sensor_driver.h"
#include "adc.h"
#include "spi_slave.h"

void ir_sensor_init(void) {
    init_adc();
}

uint16_t get_IR_sensor_data(uint8_t IR_number) {
    return adc_convert(IR_number);
}

void prepare_IR_data(void) {
    uint16_t sensorData_F = get_IR_sensor_data(0);
    uint16_t sensorData_B = get_IR_sensor_data(2);
    uint16_t sensorData_L = get_IR_sensor_data(1);
    uint16_t sensorData_R = get_IR_sensor_data(3);

    unsigned char data_F_0 = (unsigned char)((sensorData_F & 0xFF00) >> 8) + 1;
    unsigned char data_F_1 = (unsigned char)(sensorData_F & 0x00FF);
    if (data_F_1 == 0x00) data_F_1 = 0x01;

    unsigned char data_B_0 = (unsigned char)((sensorData_B & 0xFF00) >> 8) + 1;
    unsigned char data_B_1 = (unsigned char)(sensorData_B & 0x00FF);
    if (data_B_1 == 0x00) data_B_1 = 0x01;

    unsigned char data_L_0 = (unsigned char)((sensorData_L & 0xFF00) >> 8) + 1;
    unsigned char data_L_1 = (unsigned char)(sensorData_L & 0x00FF);
    if (data_L_1 == 0x00) data_L_1 = 0x01;

    unsigned char data_R_0 = (unsigned char)((sensorData_R & 0xFF00) >> 8) + 1;
    unsigned char data_R_1 = (unsigned char)(sensorData_R & 0x00FF);
    if (data_R_1 == 0x00) data_R_1 = 0x01;

    char buf[10] = {'S', ':', data_F_0, data_F_1, data_B_0, data_B_1, data_L_0,
data_L_1, data_R_0, data_R_1};
    return_message((char *)buf);
}

void get_IR_data(void) {
    return_message("S:IR_READY");
}
```

## APPENDIX C: RASPBERRYPI FIRMWARE CODE

stepper\_motor\_api.h

```
#ifndef _STEPPER_MOTOR_API_H
#define _STEPPER_MOTOR_API_H

void robotMotorModuleInit(uint8_t en_pin);
void robotGo(uint8_t speed_r, uint8_t direction_r, uint8_t speed_l, uint8_t direction_l);
void robotStop(void);
void robotGet(void);
void robotCGet(void);

#endif
```



stepper\_motor\_api.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "gpio.h"
#include "spi.h"

#include "stepper_motor_api.h"

uint8_t Motor_Module_Sel_Pin = 4;

void robotMotorModuleInit(uint8_t en_pin) {
    Motor_Module_Sel_Pin = en_pin;
    printf("The Motor Module Connected to Pin%d\n", en_pin);
}

void robotGo(uint8_t speed_r, uint8_t direction_r, uint8_t speed_l, uint8_t direction_l) {
    gpio_t spi_sel;
    spi_t spi;
    bool value;
    uint8_t buf[10] = {'M', ':', 'G', 'O', speed_r, '0'+direction_r, speed_l,
'0'+direction_l, '\n'};
    printf("shifted out: %s\n", buf);

    if (gpio_open(&spi_sel, Motor_Module_Sel_Pin, GPIO_DIR_OUT) < 0) {
        fprintf(stderr, "gpio_open(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (gpio_write(&spi_sel, !value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (spi_open(&spi, "/dev/spidev0.0", 0, 10000) < 0) {
        fprintf(stderr, "spi_open(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    if (spi_transfer(&spi, buf, buf, sizeof(buf)) < 0) {
        fprintf(stderr, "spi_transfer(): %s\n", spi_errmsg(&spi));
        exit(1);
    }
}
```

```

printf("shifted in: %s\n", buf);

spi_close(&spi);

if (gpio_write(&spi_sel, value) < 0) {
    fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
    exit(1);
}

gpio_close(&spi_sel);
}

void robotStop(void) {
    gpio_t spi_sel;
    spi_t spi;
    bool value;
    uint8_t buf[10] = {'M', ':', 'S', 'T', 'O', 'P', ' ', ' ', ' ', '\n'};
    printf("shifted out: %s\n", buf);

    if (gpio_open(&spi_sel, Motor_Module_Sel_Pin, GPIO_DIR_OUT) < 0) {
        fprintf(stderr, "gpio_open(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (gpio_write(&spi_sel, !value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (spi_open(&spi, "/dev/spidev0.0", 0, 10000) < 0) {
        fprintf(stderr, "spi_open(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    if (spi_transfer(&spi, buf, buf, sizeof(buf)) < 0) {
        fprintf(stderr, "spi_transfer(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    printf("shifted in: %s\n", buf);

    spi_close(&spi);

    if (gpio_write(&spi_sel, value) < 0) {

```

```

        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    gpio_close(&spi_sel);
}

void robotGet(void) {
    gpio_t spi_sel;
    spi_t spi;
    bool value;
    uint8_t buf[10] = {'M', ':', 'G', 'E', 'T', ',', ',', ',', ',', '\n'};
    printf("shifted out: %s\n", buf);

    if (gpio_open(&spi_sel, Motor_Module_Sel_Pin, GPIO_DIR_OUT) < 0) {
        fprintf(stderr, "gpio_open(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (gpio_write(&spi_sel, !value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (spi_open(&spi, "/dev/spidev0.0", 0, 10000) < 0) {
        fprintf(stderr, "spi_open(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    if (spi_transfer(&spi, buf, buf, sizeof(buf)) < 0) {
        fprintf(stderr, "spi_transfer(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    uint32_t rightCount = ((uint32_t)(buf[5]-1) << 24) | ((buf[4]-1) << 16) |
    ((buf[3]-1) << 8) | (buf[2]-1);
    uint32_t leftCount = ((uint32_t)(buf[9]-1) << 24) | ((buf[8]-1) << 16) | ((buf[7]-1)
    << 8) | (buf[6]-1);
    printf("shifted in: %c%cL=%ld,R=%ld\n", buf[0], buf[1], leftCount, rightCount);

    spi_close(&spi);

    if (gpio_write(&spi_sel, value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }
}

```

```

    }

    gpio_close(&spi_sel);
}

void robotCGet(void) {
    gpio_t spi_sel;
    spi_t spi;
    bool value;
    uint8_t buf[10] = {'M', ':', 'C', 'G', 'E', 'T', ' ', ' ', ' ', '\n'};
    printf("shifted out: %s\n", buf);

    if (gpio_open(&spi_sel, Motor_Module_Sel_Pin, GPIO_DIR_OUT) < 0) {
        fprintf(stderr, "gpio_open(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (gpio_write(&spi_sel, !value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (spi_open(&spi, "/dev/spidev0.0", 0, 10000) < 0) {
        fprintf(stderr, "spi_open(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    if (spi_transfer(&spi, buf, buf, sizeof(buf)) < 0) {
        fprintf(stderr, "spi_transfer(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    uint32_t leftCount = ((uint32_t)(buf[5]-1) << 24) | ((buf[4]-1) << 16) | ((buf[3]-1)
<< 8) | (buf[2]-1);
    uint32_t rightCount = ((uint32_t)(buf[9]-1) << 24) | ((buf[8]-1) << 16) |
((buf[7]-1) << 8) | (buf[6]-1);
    printf("shifted in: %c%cL=%ld,R=%ld\n", buf[0], buf[1], leftCount, rightCount);

    spi_close(&spi);

    if (gpio_write(&spi_sel, value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }
}

```

```
    gpio_close(&spi_sel);  
}
```

ir\_sensor\_api.h

```
#ifndef _STEPPER_MOTOR_API_H
#define _STEPPER_MOTOR_API_H

void IRSensorModuleInit(uint8_t en_pin);
void getIR(uint16_t* data);
void preIR(void);
void get_ir_datas(uint16_t* data);
uint16_t get_ir_data(uint8_t* IR_Sensor);

#endif
```

ir\_sensor\_api.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#include "gpio.h"
#include "spi.h"

#include "ir_sensor_api.h"

uint8_t IR_Sensor_Module_Sel_Pin = 17;

void IRSensorModuleInit(uint8_t en_pin) {
    IR_Sensor_Module_Sel_Pin = en_pin;
    printf("The IR Sensor Module Connected to Pin%d\n", en_pin);
}

void preIR(void) {
    gpio_t spi_sel;
    spi_t spi;
    bool value;
    uint8_t buf[10] = {'S', ':', 'P', 'R', 'E', ' ', 'I', 'R', ' ', '\n'};

    if (gpio_open(&spi_sel, IR_Sensor_Module_Sel_Pin, GPIO_DIR_OUT) < 0) {
        fprintf(stderr, "gpio_open(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (gpio_write(&spi_sel, !value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (spi_open(&spi, "/dev/spidev0.0", 0, 10000) < 0) {
        fprintf(stderr, "spi_open(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    if (spi_transfer(&spi, buf, buf, sizeof(buf)) < 0) {
        fprintf(stderr, "spi_transfer(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    printf("shifted in: %s\n", buf);
}
```

```

spi_close(&spi);

if (gpio_write(&spi_sel, value) < 0) {
    fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
    exit(1);
}

gpio_close(&spi_sel);
}

void getIR(uint16_t* data) {
    gpio_t spi_sel;
    spi_t spi;
    bool value;
    uint8_t buf[10] = {'S', ':', 'G', 'E', 'T', ',', 'I', 'R', ',', '\n'};

    if (gpio_open(&spi_sel, IR_Sensor_Module_Sel_Pin, GPIO_DIR_OUT) < 0) {
        fprintf(stderr, "gpio_open(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (gpio_write(&spi_sel, !value) < 0) {
        fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
        exit(1);
    }

    if (spi_open(&spi, "/dev/spidev0.0", 0, 10000) < 0) {
        fprintf(stderr, "spi_open(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    if (spi_transfer(&spi, buf, buf, sizeof(buf)) < 0) {
        fprintf(stderr, "spi_transfer(): %s\n", spi_errmsg(&spi));
        exit(1);
    }

    uint16_t dataFront = ((uint16_t)(buf[2]-1) << 8) | (buf[3]);
    uint16_t dataBack = ((uint16_t)(buf[4]-1) << 8) | (buf[5]);
    uint16_t dataLeft = ((uint16_t)(buf[6]-1) << 8) | (buf[7]);
    uint16_t dataRight = ((uint16_t)(buf[8]-1) << 8) | (buf[9]);
    printf("shifted in: %c%cF=%d,B=%d,L=%d,R=%d\n", buf[0], buf[1], dataFront,
dataBack, dataLeft, dataRight);

    data[0] = dataFront;
    data[1] = dataBack;
}

```



```

data[2] = dataLeft;
data[3] = dataRight;

spi_close(&spi);

if (gpio_write(&spi_sel, value) < 0) {
    fprintf(stderr, "gpio_write(): %s\n", gpio_errmsg(&spi_sel));
    exit(1);
}

gpio_close(&spi_sel);
}

void get_ir_datas(uint16_t* data) {
    preIR();
    //delay
    int i;
    for (i = 0; i < 50000; i++);
    getIR(data);
}

uint16_t get_ir_data(uint8_t* IR_Sensor) {
    uint16_t data[4];
    get_ir_datas(data);
    if (!strcmp(IR_Sensor, "front")) {
        return data[0];
    } else if (!strcmp(IR_Sensor, "back")) {
        return data[1];
    } else if (!strcmp(IR_Sensor, "left")) {
        return data[2];
    } else if (!strcmp(IR_Sensor, "right")) {
        return data[3];
    } else {
        return 0;
    }
}
}

```

## APPENDIX D: EXAMPLE PROJECT CODE

### Demo Project 1: Basic Control

```
main.c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "stepper_motor_api.h"
#include "ir_sensor_api.h"

int main(int argc, char const *argv[])
{
    robotMotorModuleInit(4);
    IRSensorModuleInit(17);
    printf("%s\n", argv[1]);
    if (!strcmp(argv[1], "go")) {
        if (argc > 3) {
            uint8_t speed_r = atoi(argv[2]);
            uint8_t direction_r = atoi(argv[3]);
            uint8_t speed_l = atoi(argv[4]);
            uint8_t direction_l = atoi(argv[5]);
            printf("%d %d %d %d\n", speed_r, direction_r, speed_l, direction_l);
            robotGo(speed_r, direction_r, speed_l, direction_l);
        } else {
            robotGo(50, 0, 50, 0);
        }
    } else if (!strcmp(argv[1], "stop")) {
        robotStop();
    } else if (!strcmp(argv[1], "get")) {
        robotGet();
    } else if (!strcmp(argv[1], "cget")) {
        robotCGet();
    } else if (!strcmp(argv[1], "irget")) {
        if (argc > 2) {
            printf("IR Sensor %s: %d\n", argv[2], get_ir_data(argv[2]));
        } else {
            uint16_t data[4];
            get_ir_datas(data);
            printf("F:%d B:%d L:%d R:%d\n", data[0], data[1], data[2],
data[3]);
        }
    }
}
```

```
        }  
    }  
    return 0;  
}
```

Makefile

CC=gcc

CFLAGS=-I/home/pi/c-periphery-master/src

LIBS = -lm

periphery = /home/pi/c-periphery-master/periphery.a

FILE = main.c stepper\_motor\_api.c ir\_sensor\_api.c

TARGET = robot

all: \$(TARGET)

robot: \$(FILE)

\$(CC) -o \$@ \$^ \$(periphery) \$(CFLAGS) \$(LIBS)

clean:

rm \$(TARGET)

## Demo Project 2: Obstacle Avoidance

```
stop_before_obstacle.c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "gpio.h"
#include "spi.h"

#include "stepper_motor_api.h"
#include "ir_sensor_api.h"
#include "ir_functions.h"

int main(int argc, char const *argv[])
{
    uint16_t frontLimit = Calculate_Front_IR_Value(10);
    uint16_t sensorData[4];
    int i;

    printf("Front Limit is %d\n", frontLimit);

    robotMotorModuleInit(17);
    IRSensorModuleInit(4);
    printf("Obstacle Avoidance project start!\n");

    robotGo(20,1,20,1);

    for (i = 0; i < 50000; i++);
    get_ir_datas(sensorData);
    while (sensorData[0] < frontLimit) {
        for (i = 0; i < 50000; i++);
        get_ir_datas(sensorData);
    }
    for (i = 0; i < 50000; i++);
    robotStop();
    for (i = 0; i < 50000; i++);
    robotGet();

    return 0;
}
```

```

shy_avoid.c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "gpio.h"
#include "spi.h"

#include "stepper_motor_api.h"
#include "ir_sensor_api.h"
#include "ir_functions.h"

int main(int argc, char const *argv[])
{
    uint16_t frontLimit = Calculate_Front_IR_Value(10);
    uint16_t backLimit = Calculate_Back_IR_Value(10);
    uint16_t sensorData[4];
    int i;
    uint8_t status = 0;

    printf("Front Limit is %d\n", frontLimit);
    printf("Back Limit is %d\n", backLimit);

    robotMotorModuleInit(17);
    IRSensorModuleInit(4);
    printf("Obstacle Avoidance project start!\n");

    while(1) {
        get_ir_datas(sensorData);
        for (i = 0; i < 50000; i++);
        if (sensorData[0] > frontLimit && sensorData[1] < backLimit) {
            if (status != 'F') {
                robotGo(20,0,20,0);
                for (i = 0; i < 50000; i++);
                status = 'F';
            }
        } else if (sensorData[0] < frontLimit && sensorData[1] > backLimit) {
            if (status != 'B') {
                robotGo(20,1,20,1);
                for (i = 0; i < 50000; i++);
                status = 'B';
            }
        } else {

```

```
        robotStop();  
        for (i = 0; i < 50000; i++)  
    }  
}  
return 0;  
}
```

main.c for obstacle avoid

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
```

```
#include "gpio.h"
#include "spi.h"
```

```
#include "stepper_motor_api.h"
#include "ir_sensor_api.h"
#include "ir_functions.h"
```

```
uint8_t ObstacleAvoid(uint16_t frontLimit, uint16_t* sensorData, uint8_t states);
```

```
int main(int argc, char const *argv[])
```

```
{
    uint16_t frontLimit = Calculate_Front_IR_Value(10);
    uint16_t sensorData[4];
    int i;
    uint8_t states = 0;

    printf("Front Limit is %d\n", frontLimit);

    robotMotorModuleInit(4);
    IRSensorModuleInit(17);
    printf("Obstacle Avoidance project start!\n");

    while(1) {
        get_ir_datas(sensorData);
        for(i = 0; i < 50000; i++);
        states = ObstacleAvoid(frontLimit, sensorData, states);
    }

    return 0;
}
```

```
uint8_t ObstacleAvoid(uint16_t frontLimit, uint16_t* sensorData, uint8_t states) {
```

```
    int i;
    uint8_t nextStates = states;

    switch (states) {
        case 0:
            robotGo(30,1,30,1);
            for (i = 0; i < 50000; i++);
```



```

        nextStates = 1;
        break;
    case 1:
        if (sensorData[0] > frontLimit) {
            robotStop();
            for (i = 0; i < 50000; i++);
            nextStates = 2;
        }
        break;
    case 2:
        robotGo(30,0,30,1);//turn right
        for (i = 0; i < 50000; i++);
        nextStates = 3;
        break;
    case 3:
        if (sensorData[0] < frontLimit) {
            robotStop();
            for (i = 0; i < 50000; i++);
            nextStates = 0;
        }
        break;
    default:
        robotStop();
        for (i = 0; i < 50000; i++);
        break;
}

return nextStates;
}

```

### Makefile

CC=gcc

CFLAGS=-I/home/pi/c-periphery-master/src

LIBS = -lm

periphery = /home/pi/c-periphery-master/periphery.a

FILE\_SBO = stop\_before\_obstacle.c stepper\_motor\_api.c ir\_functions.c ir\_sensor\_api.c

FILE\_SA = shy\_avoid.c stepper\_motor\_api.c ir\_functions.c ir\_sensor\_api.c

FILE\_OA = main.c stepper\_motor\_api.c ir\_functions.c ir\_sensor\_api.c

TARGET = StopBeforeObstacle ShyAvoid ObstacleAvoid

all: \$(TARGET)

StopBeforeObstacle: \$(FILE\_SBO)

\$(CC) -o \$@ \$^ \$(periphery) \$(CFLAGS) \$(LIBS)

ShyAvoid: \$(FILE\_SA)

\$(CC) -o \$@ \$^ \$(periphery) \$(CFLAGS) \$(LIBS)

ObstacleAvoid: \$(FILE\_OA)

\$(CC) -o \$@ \$^ \$(periphery) \$(CFLAGS) \$(LIBS)

clean:

rm \$(TARGET)

## Demo Project 3: Wall Following

```
main.c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

#include "gpio.h"
#include "spi.h"

#include "stepper_motor_api.h"
#include "ir_sensor_api.h"
#include "ir_functions.h"

#include "pi_robot_headers.h"

#define MODE_FOLLOW_LEFT_WALL 0
#define MODE_FOLLOW_RIGHT_WALL 1
#define MODE_IN_THE_MIDDLE_OF_WALLS 2

uint8_t Kp = 2;
struct Motors motors;

uint8_t LeftWallFollowing(struct SensorLimits IRLimits, uint16_t* sensorData, uint8_t
states);
void P_Controller(struct SensorLimits IRLimits, uint16_t* sensorData, uint8_t mode);

int main(int argc, char const *argv[])
{
    struct SensorLimits IRLimits;
    IRLimits.frontLimit = Calculate_Front_IR_Value(10);
    IRLimits.leftLimit = Calculate_Left_IR_Value(5);
    IRLimits.rightLimit = Calculate_Right_IR_Value(5);
    uint16_t sensorData[4];
    int i;
    uint8_t states = 0;

    printf("Front Limit is %d\n", IRLimits.frontLimit);
    printf("Left Limit is %d\n", IRLimits.leftLimit);
    printf("Right Limit is %d\n", IRLimits.rightLimit);

    motors.speed_l = 30;
    motors.speed_r = 30;
    motors.direction_l = 1;
```

```

motors.direction_r = 1;

robotMotorModuleInit(4);
IRSensorModuleInit(17);

while(1) {
    get_ir_datas(sensorData);
    for(i = 0; i < 50000; i++);
    states = LeftWallFollowing(IRLimits, sensorData, states);
}

return 0;
}

```

```

uint8_t LeftWallFollowing(struct SensorLimits IRLimits, uint16_t* sensorData, uint8_t
states) {
    int i;
    uint8_t nextStates = states;

    switch (states) {
        case 0:
            robotGo(motors.speed_r, motors.direction_r, motors.speed_l,
motors.direction_l);
            for (i = 0; i < 50000; i++);
            nextStates = 1;
            break;
        case 1:
            if (sensorData[0] > IRLimits.frontLimit) {
                robotStop();
                for (i = 0; i < 50000; i++);
                nextStates = 2;
            } else if (sensorData[2] < Calculate_Left_IR_Value(20)) {
                robotStop();
                for (i = 0; i < 50000; i++);
                nextStates = 4;
            } else {
                P_Controller(IRLimits, sensorData,
MODE_FOLLOW_LEFT_WALL);
                robotGo(motors.speed_r, motors.direction_r,
motors.speed_l, motors.direction_l);
                for (i = 0; i < 50000; i++);
                nextStates = 1;
            }
            break;
        case 2:

```

```

        robotGo(30,0,30,1);
        for (i = 0; i < 50000; i++);
        nextStates = 3;
        break;
    case 3:
        if (sensorData[0] < IRLimits.frontLimit && sensorData[2] >
Calculate_Left_IR_Value(9)) {
            robotStop();
            for (i = 0; i < 100000; i++);
            nextStates = 0;
            motors.speed_l = 30;
            motors.speed_r = 30;
            motors.direction_l = 1;
            motors.direction_r = 1;
        } else {
            nextStates = 3;
        }
        break;
    case 4:
        robotGo(50,1,15,1);
        for (i = 0; i < 50000; i++);
        nextStates = 5;
        break;
    case 5:
        if (sensorData[0] > Calculate_Left_IR_Value(8)) {
            robotStop();
            for (i = 0; i < 50000; i++);
            nextStates = 6;
        } else {
            nextStates = 5;
        }
        break;
    case 6:
        robotGo(30,0,30,1);
        for (i = 0; i < 50000; i++);
        nextStates = 7;
        break;
    case 7:
        if (sensorData[0] < IRLimits.frontLimit && sensorData[2] >
Calculate_Left_IR_Value(5)) {
            robotStop();
            for (i = 0; i < 100000; i++);
            nextStates = 1;
            motors.speed_l = 30;
            motors.speed_r = 30;

```

```

        motors.direction_l = 1;
        motors.direction_r = 1;
    } else {
        nextStates = 7;
    }
    break;
default:
    robotStop();
    for (i = 0; i < 50000; i++);
    break;
}

return nextStates;
}

void P_Controller(struct SensorLimits IRLimits, uint16_t* sensorData, uint8_t mode) {
    int cur_error;
    double cur_error_inch;
    switch (mode) {
        case MODE_FOLLOW_LEFT_WALL:
            cur_error = IRLimits.leftLimit - sensorData[2];
            cur_error_inch =
Calculate_Left_Inch_Value((double)sensorData[2]) - 10.;
            motors.speed_l = (uint8_t)(30. - cur_error_inch * ((double)Kp));
            motors.speed_r = 30;
            printf("the error is %d, and %f inch\n", cur_error, cur_error_inch);
            printf("the left speed is %d\n", motors.speed_l);
            printf("the right speed is %d\n", motors.speed_r);
            break;
    }
}
}

```

```
pi_robot_headers.h
#ifndef _PI_ROBOT_HEADERS_H
#define _PI_ROBOT_HEADERS_H

struct Motors {
    uint8_t speed_l;
    uint8_t speed_r;
    uint8_t direction_l;
    uint8_t direction_r;
};

struct SensorLimits {
    uint16_t frontLimit;
    uint16_t backLimit;
    uint16_t leftLimit;
    uint16_t rightLimit;
};

#endif
```

Makefile

CC=gcc

CFLAGS=-I/home/pi/c-periphery-master/src

LIBS = -lm

periphery = /home/pi/c-periphery-master/periphery.a

FILE\_WF = main.c stepper\_motor\_api.c ir\_functions.c ir\_sensor\_api.c

TARGET = WallFollowing

all: \$(TARGET)

WallFollowing: \$(FILE\_WF)

\$(CC) -o \$@ \$^ \$(periphery) \$(CFLAGS) \$(LIBS)

clean:

rm \$(TARGET)



## APPENDIX E: LIST OF DEMO VIDEOS

Demo of Basic Control: <https://www.youtube.com/watch?v=72nGzVvk8Yg>

Demo of Obstacle Avoidance: <https://www.youtube.com/watch?v=Emuj7UCeCBA>

Demo of Wall Following (inner corner): <https://www.youtube.com/watch?v=PRldmvRWyXk>

Demo of Wall Following (long wall): <https://www.youtube.com/watch?v=DnVG7MBfuXA>

Demo of Wall Following (outer corner): <https://www.youtube.com/watch?v=OIhhXDOD0mM>

## APPENDIX F: USER'S MANUAL

The modular educational robotics platform has been built by Wei Zhen on the Raspberry Pi 2 platform, which runs the 32-bit Embedded Linux System “Raspbian”, several modules, which designed by 16-bit microcontroller and used to control all of the features this robot has, and some other mechanical and electrical parts, such as frame, motors, and sensors. The modules are used to control all of the motors and sensors, and the Raspberry Pi 2 is used to control all of the modules called by the Central Controller. Modules and Central Controller are connected by SPI. Central Controller is the master, and modules are slaves.

The Central Controller uses WiFi technology to connect and control it. It runs a DHCP server under Linux System, which makes it easier to find and connect to computers. And the 32-bit Embedded Linux System gives users more options about programming languages. Users can use any kind of language they like to program and control the robot. For this menu only the C library for Central Controller is included, but with communication API and web sources documentations, users can easily program the robot in another language, such as Python and Lua.

Modules are designed to control all of the motions and feedbacks which means all of the motors and sensors. Currently, all of the modules are designed under TI's 16-bit microcontroller MSP430G2553, and only Stepper Motor Controlling Module, IR Sensor Module, and Custom Module are included in the design. However, with this documentation, users can easily figure out the way to design their own modules to control everything they want with or without MSP430G2553.

The examples and libraries are stored in github, which address is [git@github.com:weizhen1883/Modular-Robot-Design-MastersThesis.git](https://github.com/weizhen1883/Modular-Robot-Design-MastersThesis.git), user can easily download all of the code and hardware designs of the robot. They can use it as examples for their own design, too.

### Wire the Robot

1. Connect power cables to the base board. Shown in figure F-1.

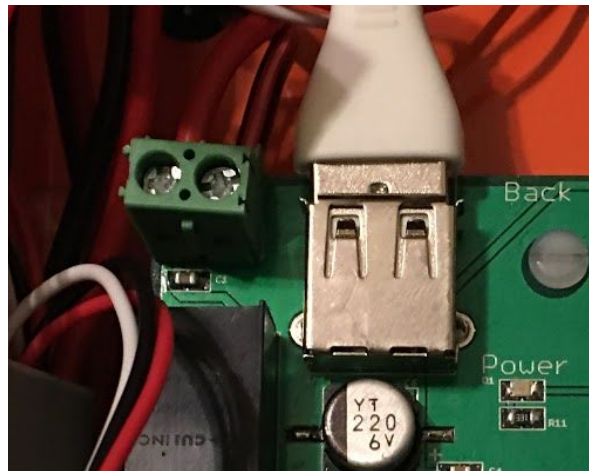


Figure F-1: Wire Power cables

2. Connect stepper motor connectors to the base board. Shown in Figure F-2.

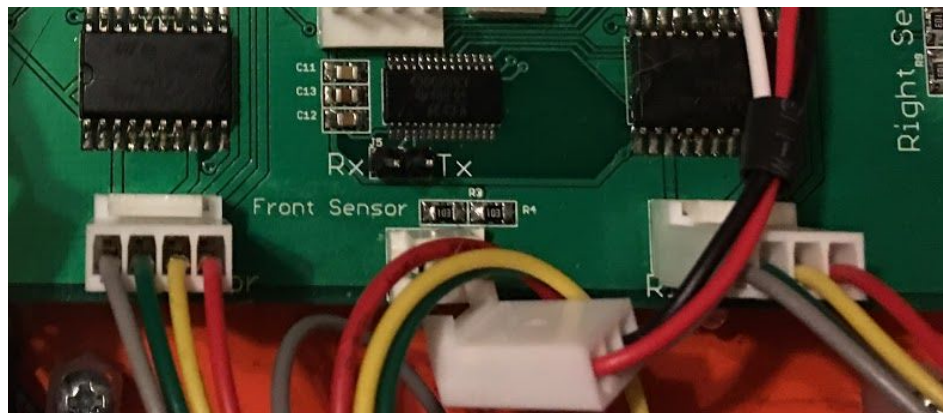


Figure F-2: Wire Stepper Motors

3. Connect IR sensors and to the base board and wire the cable who connected the base board to central controller. Shown in Figure F-3.

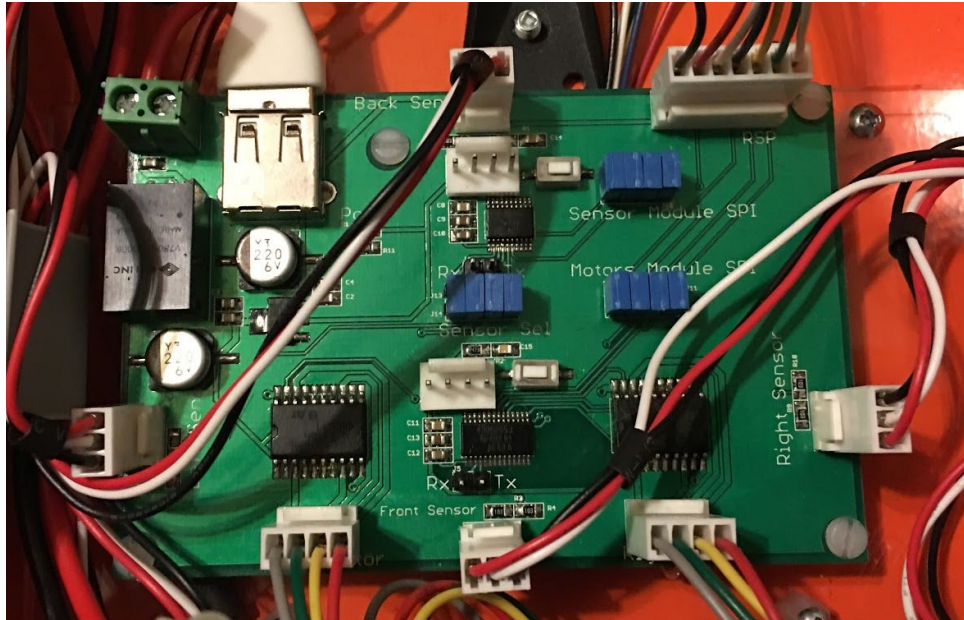


Figure F-3: IR Sensors and Cable connections

4. Connect the power switch with battery. Shown in Figure F-4.



Figure F-4: Power the Battery

5. Clean up the wires in the base. Shown in Figure F-5.

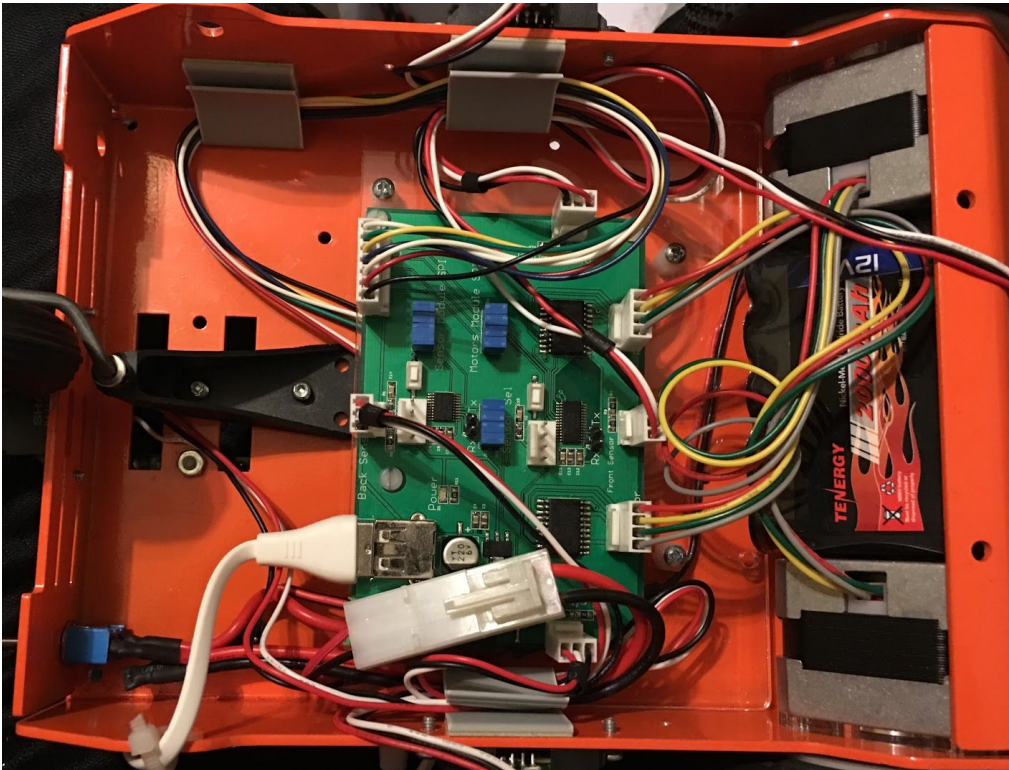


Figure F-5: Clean the Wires

6. Mount the Raspberry Pi on the top, and connect the custom module. Shown in Figure F-6.

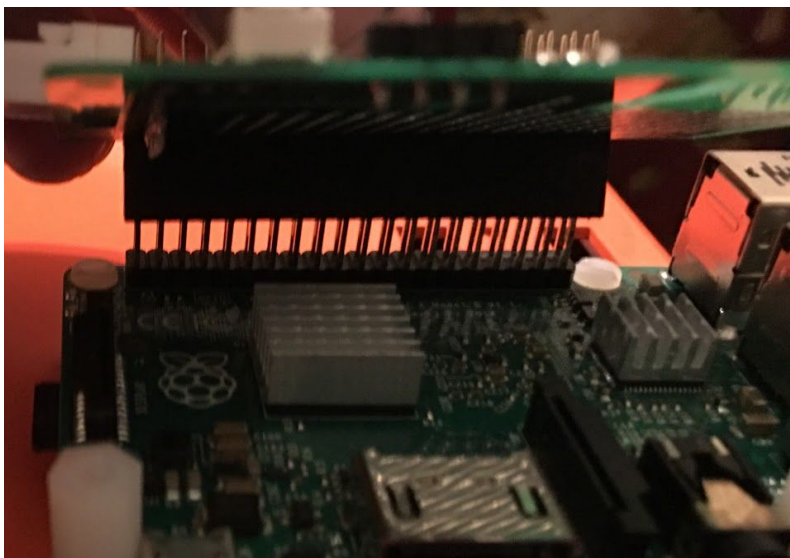


Figure F-6: Connect the custom module with Raspberry Pi

7. Power the Raspberry Pi and done all of the wiring works. Shown in Figure F-7.

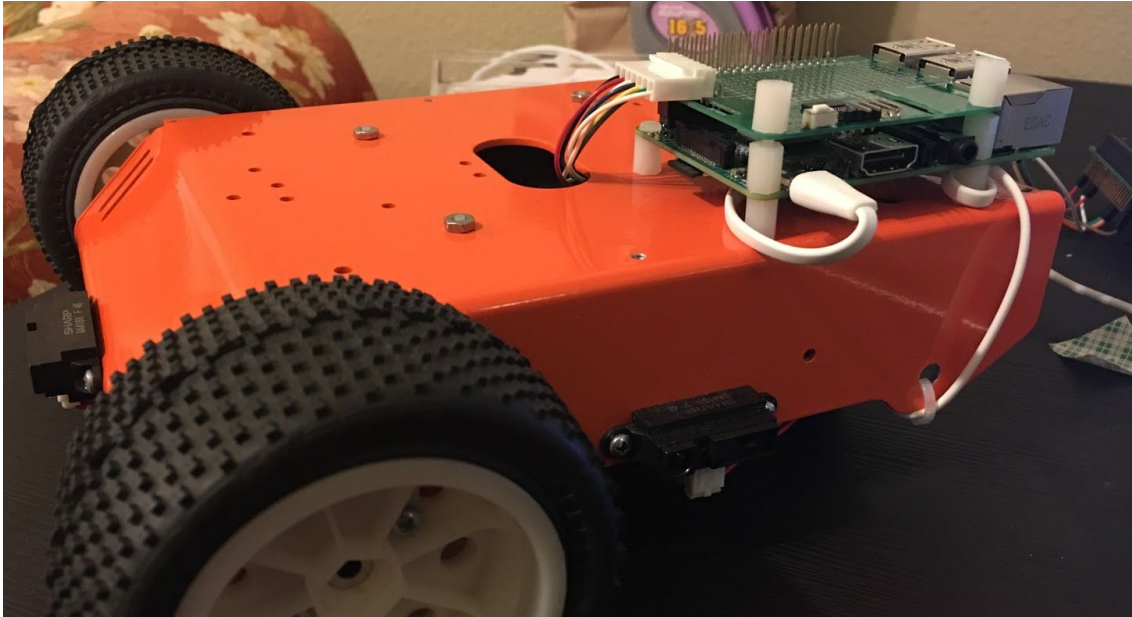


Figure F-7: Final Looking of the Robot

### Connect to the Robot

1. Turn on the power of robot
2. Find and connect to WiFi whose name is Pi\_Robot\_AP and password is RaspberryPiRobot
3. The robot has been connected
4. Use Putty or Terminal to ssh into robot by
  - a. username: pi
  - b. password: 1qaz2wsx
  - c. ip address: 192.168.42.1
5. Then you can take control of robot. Everything you need to do is just like using the Linux OS.
6. (Options) Type the 192.168.42.1 in web browsers, and controller GUI will show up

### Getting Started (For Windows Users)

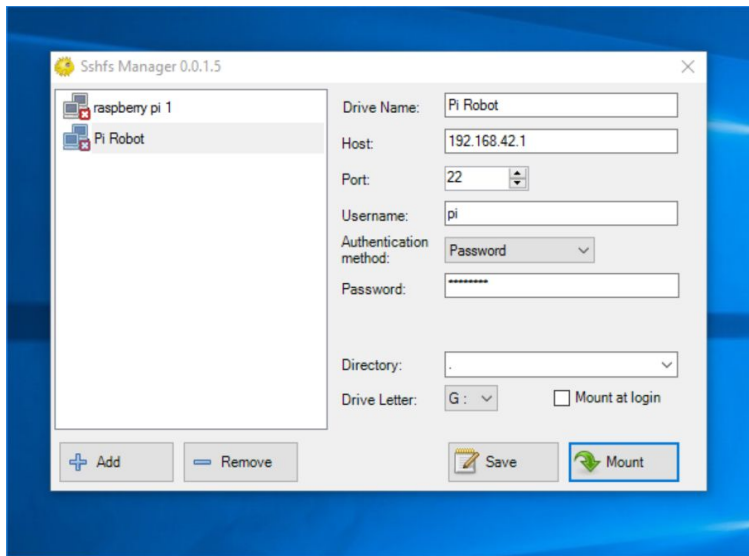
Assume that a Linux or Unix user should familiar with bash command lines and how to use ssh, this tutorial is for Windows Users who do not familiar with bash command lines and ssh.

After connected the Pi\_Robot\_AP, the user should do the following stapes.

1. Installing the sshfs follows tutorial in DigitalOcean

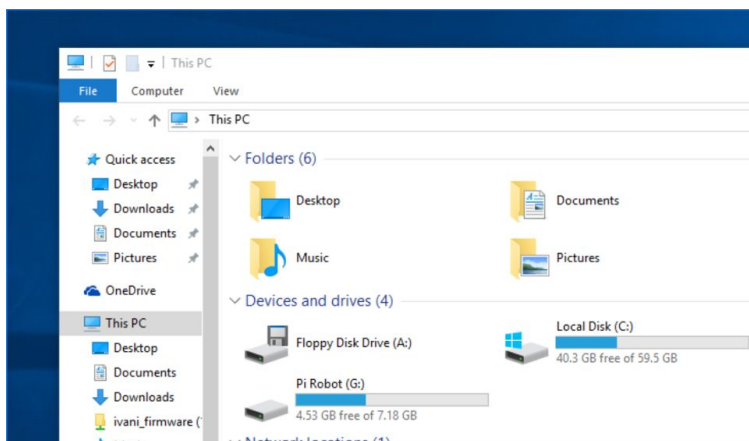
<https://www.digitalocean.com/community/tutorials/how-to-use-sshfs-to-mount-remote-file-systems-over-ssh>.

2. Mount the Robot file system by using sshfs manager.

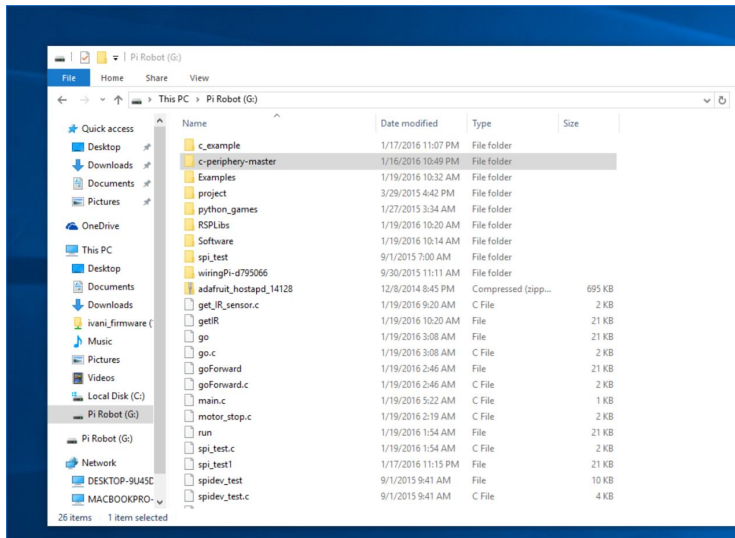


After type all the necessary information into the sshfs manager, click the “Mount” to mount robot directory into your Windows.

3. Then open “This PC”, the robot directory driver named “Pi Robot” will show up.

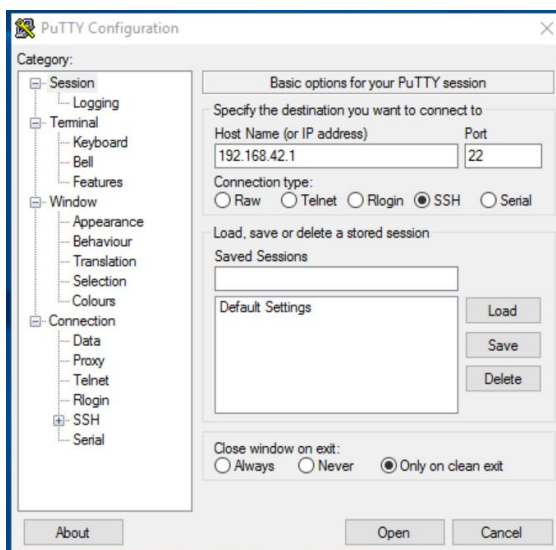


4. Double click the “Pi Robot” driver to open the robot directory.



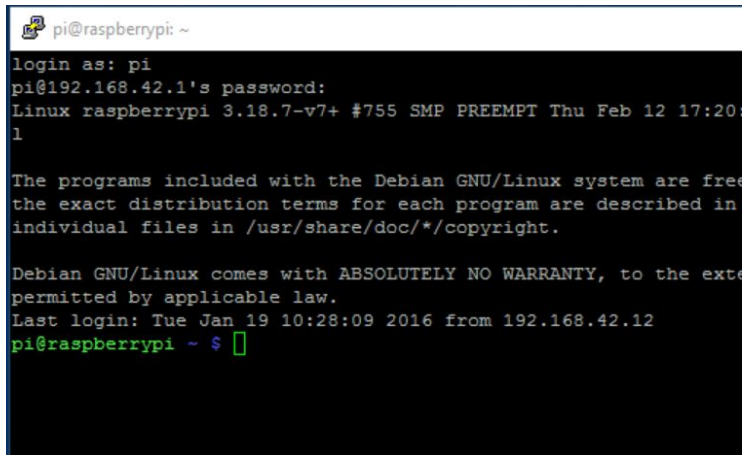
5. Then user can copy/delete the code, write their own code into this directory as same as they did in Windows.
6. Make sure the “RSPLibs” starting project is already inside the directory, if not download from github,

<https://github.com/weizhen1883/Modular-Robot-Design-MastersThesis/tree/master/Firmware/RSPLibs>. Then use the putty to connect to the robot.





- Click the “Open”, it will open an new window which will require user enter the username and password.

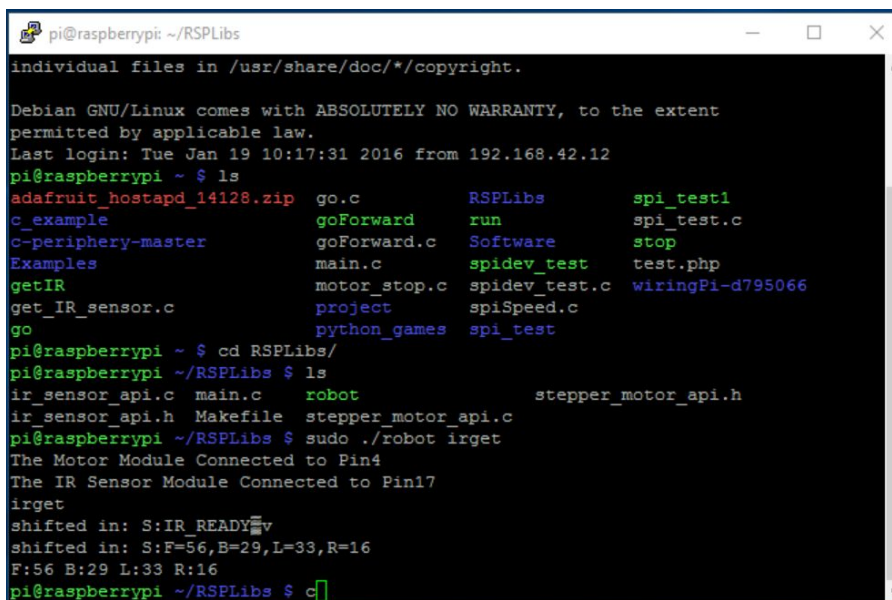


```
pi@raspberrypi: ~
login as: pi
pi@192.168.42.1's password:
Linux raspberrypi 3.18.7-v7+ #755 SMP PREEMPT Thu Feb 12 17:20:
l

The programs included with the Debian GNU/Linux system are free
the exact distribution terms for each program are described in
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the exte
permitted by applicable law.
Last login: Tue Jan 19 10:28:09 2016 from 192.168.42.12
pi@raspberrypi ~ $
```

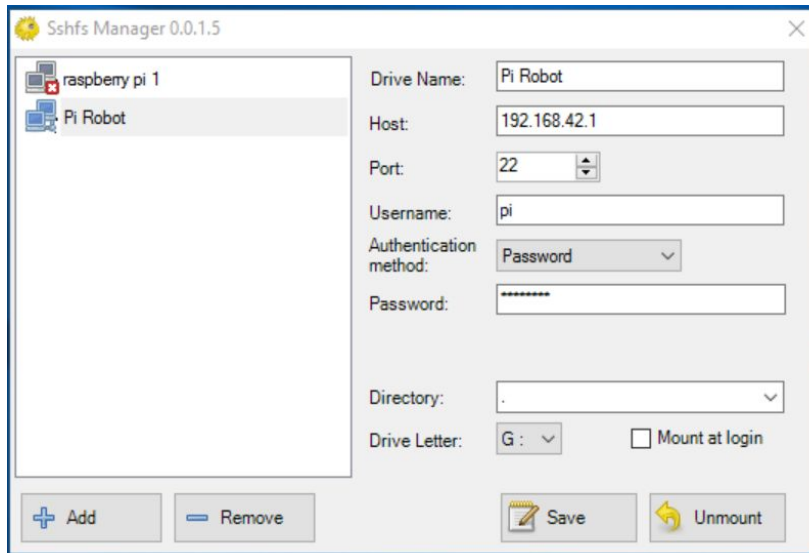
- Now the robot has been connected and ready to use. In Linux “ls” used to show everything inside the current direction, and “cd <folder name>” used to get into a folder. After get into the “RSPLibs” folder, user can run “sudo ./robot <command>” to control the robot. For example, “go” ask the robot go, “stop” ask the robot stop, and “irget” get the IR data.



```
pi@raspberrypi: ~/RSPLibs
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jan 19 10:17:31 2016 from 192.168.42.12
pi@raspberrypi ~ $ ls
adafruit_hostapd_14128.zip  go.c          RSPLibs       spi_test1
c_example                 goForward    run           spi_test.c
c-periphery-master       goForward.c  Software      stop
Examples                 main.c       spidev_test  test.php
getIR                    motor_stop.c spidev_test.c wiringPi-d795066
get_IR_sensor.c         project      spiSpeed.c
go                       python_games spi_test
pi@raspberrypi ~ $ cd RSPLibs/
pi@raspberrypi ~/RSPLibs $ ls
ir_sensor_api.c  main.c  robot          stepper_motor_api.h
ir_sensor_api.h  Makefile  stepper_motor_api.c
pi@raspberrypi ~/RSPLibs $ sudo ./robot irget
The Motor Module Connected to Pin4
The IR Sensor Module Connected to Pin17
irget
shifted in: S:IR_READY
shifted in: S:F=56,B=29,L=33,R=16
F:56 B:29 L:33 R:16
pi@raspberrypi ~/RSPLibs $ c
```

- After finishing using the robot, just unmount the robot from the Windows machine by click the “Unmount” in sshfs manager.



### Program the Robot

The C Library of robot in the Central Controller includes two apis, `stepper_motor_api` and `ir_sensor_api`, and all the functions are listed in Table E-1 for Stepper Motor API and Table E-2 for IR Sensor API. When using Stepper Motor AP, users need to include the `stepper_motor_api.h`. Include the `ir_sensor_api.h` for using IR Sensor API.

**Table E-1: Stepper Motor Control API**

Functions	Descriptions
<code>void robotMotorModuleInit (uint8_t en_pin)</code>	The function uses to init the robot stepper motor module, which has an input pin number for module selections pin. The default pin number is GPIO 17, so the input is 17.
<code>void robotGo (uint8_t speed_r, uint8_t direction_r, uint8_t speed_l, uint8_t direction_l)</code>	This function uses to set the speed and directions of the robot’s motors. It has four inputs, <code>speed_r</code> , <code>direction_r</code> , <code>speed_l</code> , and <code>direction_l</code> . The speed is between 0 to 50 rpm. The direction is 0 or 1, 1 means go forward and 0 means backward.
<code>void robotStop (void)</code>	This function uses to make the robot stop.
<code>void robotGet (uint32_t* motorCycles)</code>	This function uses to get the motorCycle value, but must call after

	the robotStop function called.
void robotCGet (uint32_t* motorCycles)	This function uses to get the motorCycle value during robot is running.

**Table E-2: IR Sensor API**

Functions	Descriptions
void IRSensorModuleInit (uint8_t en_pin)	This function uses to init the robot ir sensor module, which has an input pin number for module selections pin. The default pin number is GPIO 4, so the input is 4.
void getIR (uint16_t* data)	This function uses to get all four IR data back. It is an internal function which is called by get_ir_datas. The input for getIR is an integer array.
void preIR (void)	This function uses to ask the ir sensor module to prepare the ir data. It is an internal function which is called by get_ir_datas. The input for preIR is an integer array.
void get_ir_datas (uint16_t* data)	This function uses to get ir datas. The input is an integer array pointer, and it is an output too.
uint16_t get_ir_data (uint8_t* IR_Sensor)	This function uses to get one ir data back. The input is a sensor name, it can be “left”, “right”, “front”, and “back”. It returns an integer which is the value of this ir sensor.

When the user is programming the robot, they should call the init function first, and then call the other functions. The 5000 cycles delay is necessary between calling two of these functions.

A makefile is necessary to compile the robot programs. The Code E-1 shows the format of a simple makefile for C API.

**Code E-1: simple makefile for C API**

```

CC=gcc
CFLAGS=-I/home/pi/c-periphery-master/src
LIBS = -lm
periphery = /home/pi/c-periphery-master/periphery.a

API_FILES = <*_api.c files>
FILES = <all necessary .c files>

TARGET = <list of project names>

all: $(TARGET)

projectName: $(FILES) $(API_FILES)

```

```
$(CC) -o $@ $^ $(periphery) $(CFLAGS) $(LIBS)

clean:
    rm $(TARGET)
```

In the makefile, the `c-peripher-master` is the location where the SPI and GPIO libraries are stored. “The `c-periphery` is a set of C wrapper functions for GPIO, SPI, I2C, MMIO, and Serial peripheral I/O interface access in userspace Linux. The `c-periphery` wrappers simplify and consolidate the native Linux APIs to these interfaces. `c-periphery` is useful in embedded Linux environments (including BeagleBone, Raspberry Pi, etc. platforms) for interfacing with external peripherals. `c-periphery` is re-entrant, uses static allocations, has no dependencies outside the standard C library and Linux, compiles into a static library for easy integration with other projects, and is MIT licensed” [8]. For more documentation, the github project `c-peripher` can be checked out on: <https://github.com/vsergeev/c-periphery.git>. There are another two peripher projects in Python and Lua, whose links can be found under the README.md file in this project.

After using makefile to compile the code, the files with project name are generated. Then the command “\$ sudo ./<project name>” should be used to run the program. The program must be run with sudo, due to the permission issue.

### Communication API Between Central Controller and Modules

The Central Controller controls the robot by sending and getting the data sent through the SPI communication bus. It sends and receives a 10 bits message buf in the format:

X(module identity):XXXXXXXX(message)\n (Format E-1)

Such as “M:GO2121 \n” means robot go at speed 50, direction forward, for both motors, where the “M” means motor, “GO” means action, and the value of ‘2’ is 50 so the speed is 50. The Communication APIs for the stepper motor module and the IR sensor module are in Table E-3 and Table E-4. The format of the 10-bit message buf is built by a 1 bit module identity character, like ‘M’ for Motor module and ‘S’ for IR sensor module, ‘:’, which separates the module identity and message. Then a 7-bit message and a new line character ‘\n’. The message has 7 bits but it should not always be 7 bits, if the useful message is less than 7 bits, the space must be used to fill the message buf. After the commands are sent to the modules, the modules will return a 10-bit message buf to the Central Control. The format of returning message is

X(module identity):XXXXXXXX(message) (Format E-2)

The returning messages are also listing in Table E-3 and E-4.

**Table E-3: Stepper Motor Communication APIs**

Commands	Descriptions	Returns
M:GO<speed_r><direction_r><speed_l><direction_l>	Ask the robot go with specific speed and direction. The speed is a uint8_t value with range 0 to 50. And the direction is character ‘0’ or ‘1’.	M:RUNNING.
M:STOP	Ask the robot stop	N/A
M:GET	Ask the robot stop, and return the current motor cycle counts	M:<4-bit right count><4-bit left count>
M:CGET	Ask the robot return the current motor cycle and without stop the robot	M:<4-bit right count><4-bit left count>

**Table E-4: IR Sensor Communication APIs**

Commands	Descriptions	Returns
S:GET IR	Ask to get the IR data	S:<2-bit front data><2-bit back data><2-bit left data><2-bit right data>
S:PRE IR	Ask the robot to prepare the IR data, and make it ready for ir get command.	N/A

## Plug and Play Module Design Menu

This robot platform is a plug and play platform, and it allows user to design their own modules, which can be easily plugged in to work with the platform . If user wants to keep using MSP430G2553 to design their own modules, they can follow the design of Custom Module and make their own design very easily.

First, they should delete the J2 in the schematic of Custom Module, which shows in Figure E-8. The J2 only works to connect the robot base into the SPI bus. The Custom Module already has this feature, so it is no longer needed.

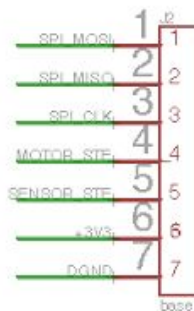


Figure E-8: SPI Bus Connector for Robot Base

Second, they need to change the SPI STE pin for MSP430G2553 on J1, which shows in Figure E-9. J1 is the connector which will connect to the Raspberry Pi. As showing, GPIO 17 is for the motor module, GPIO 04 is for the IR sensor module, and GPIO 18 is for the Custom Module. Therefore, the user should to change SPI\_STE pin to one of any other GPIOs. Be sure to keep the J1 connector, because it is the most important part to making the platform plug and play.

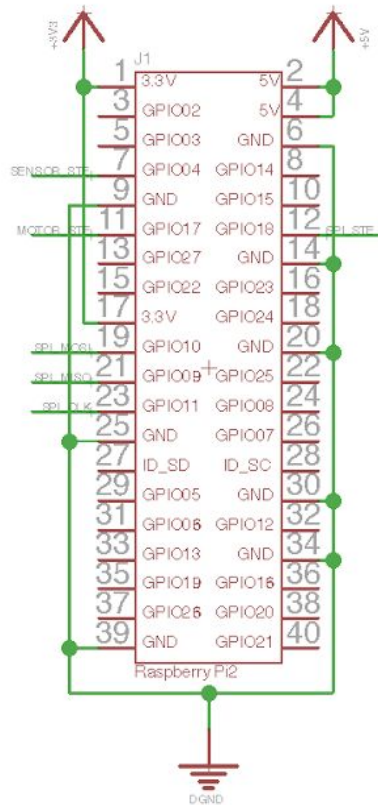


Figure E-9: Raspberry Pi 2 Connector

And then, the user can delete everything else except the Raspberry Pi 2 connector and MSP430G2553 chip groups. Now they can make their own designs like adding sensors, screens, etc.

For users who do not want to use MSP430G2553 Microcontroller, basically they will do the same thing first, and then just make sure that microcontroller has 4-wire SPI port and replace the MSP430G2553 with that microcontroller, and connect the SPI pins with the Raspberry Pi 2 Connector.

## APPENDIX G: SIMPLE EDUCATIONAL PROJECT

### Simple Lab: Obstacle Avoidance

#### Purpose:

An essential characteristic of an autonomous robot is the ability to navigate in an environment safely. The purpose of this lab is to develop obstacle avoidance behaviors by using the modular educational robotics platform. After this lab, students should get ideas about how to develop this robot by using the given hardware and firmware libraries.

#### Objectives:

At the conclusion of this lab, the student should be able to:

- Write obstacle avoidance behaviors on the modular educational robotics platform
- Move the robot safely in an environment with obstacles
- Program the robot with given library and APIs
- Design their own module

#### Equipment:

- The Modular Educational Robotics Platform
- Custom Module
- Ultrasound Sensors

#### Theory:

The Modular Educational Robotics Platform has a stepper motor module, which controls two stepper motors, and an IR sensor module, which controls four Sharp GP2Y0D810Z0F contact sensor, which is a reflectance sensor that emits and detects infrared light. The



specifications state that the Sharp contact sensors measure between 1.5” and 12”. The IR sensor will return the analog value and the IR sensor module will return the analog readings that is proportional to the distance to an object. In the library, the `ir_sensor_api` should be used to make it easy to get IR data. And the `stepper_motor_api` will be helpful for controlling the motions part of the robot.

### Part 1: Range IR Sensors

In order to determine the specific characteristics of the range sensors on the robot, you should take several measurements on each sensor to correlate the measured distance with the actual distance to an obstacle. You will need to use these data to figure out the relationship between distances and IR sensor analog values. You should write code to transfer the distance to analog value or analog value to distance. The best way to help you find the function for relationship between distances and analog values would be use a data table and perform an error analysis. (see Table F-1)

Table F-1: Sensor Calibration Data

Distance (inch)	IR Front	IR Back	IR Left	IR Right
1				
...				
15				

### Part 2: Avoid Obstacle

The obstacle avoidance require the robot to do some behaviors like stop and make a turn when the robot gets to the obstacle. The goal for this project is make the robot go past the

obstacle when it gets to the obstacle. For this part, it require to program for three small milestones.

Milestone 1: the robot goes forward, and stops when it gets to the obstacle.

Milestone 2: the robot goes forward, and makes a turn to avoid the obstacle when it gets to the obstacle.

Milestone 3: the robot goes forward, and bypasses the obstacle when it gets to the obstacle.

Notice: The Obstacle is a cube whose width, length, and height are 10 inches.

### Part 3: Ultrasound Sensor

Use the Custom Module to make the Ultrasound Sensors work and redo the obstacle avoidance project. For this part, you need to wire and mount the ultrasound sensor first. And then, program the module by writing your own firmware following the example msp430 programs. At the end, make the robot do the same things as IR sensor did.

In this part, you need to

- Wire and mount the ultrasound sensor
- Write custom firmware, and make it communicate with central controller
- Write functions on central controller to control the custom module
- Range sensors
- Write programs to make the robot avoid obstacle