CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

THE DESIGN OF A SINGLE-LEGGED ROBOT

A thesis submitted in partial fulfillment of the requirements
For the degree of Master of Science in Electrical Engineering

By

Shari Eskenas

May 2012

The thesis of Shari Eskenas is approved:

Dr. Ronald W. Mehler	Date
Dr. Somnath Chattopadhyay	Date
Dr. Nagi El Naga, Chair	Date

California State University, Northridge

Table of Contents

Signature Page ii
List of Figures v
List of Tables vii
Abstract viii
Chapter 1: Overview 1
1.1 Introduction 1
1.2 Objective 1
1.3 Project Outline 4
Chapter 2: System Integration 6
2.1 Components 6
2.2 The Algorithm 8
Chapter 3: Mechanical Design 11
3.1 Designing for Stability 11
3.2 Vertical Axis Rotational Design 13
3.3 Construction Material Choices 15
Chapter 4: Hardware Design 16
4.1 Wixel Microcontroller Module 16
4.2 HS-805BB Hitec Servo 16
4.3 Sharp GP2Y0A21YK Infrared Proximity Sensors 17
4.4 298:1 Micro Metal Gearmotors and Motor Driver 1A Dual TB6612FNG 18
4.5 MMA7341L 3-Axis Accelerometer ±3/11g 18
4.6 Rechargeable Power Supplies and High Current Regulator 18
Chapter 5: Component Information 20
5.1 USB Wireless Microcontroller Module 21

5.2 HS-805BB Hitec Servo 22
5.3 Sharp GP2Y0A21YK Infrared Proximity Sensor 23
5.4 298:1 Micro Metal Gearmotor 25
5.5 Motor Driver 1A Dual TB6612FNG 25
5.6 MMA7341L 3-Axis Accelerometer ±3/11g 29
5.7 Coin Cell Rechargeable Battery and LiPo Charger 30
5.8 NiMH Power Pack 31
5.9 LT1528 High Current Voltage Regulator 31
Chapter 6: Programming 33
6.1 Development Environment 33
6.2 Pulse Width Modulation 34
6.3 Accelerometer Readings 37
6.4 Forward Movement Logic 37
6.5 Vertical Axis Rotation Logic 38
Chapter 7: Testing and Construction 40
7.1 Electronics and Logic Testing 40
7.2 Torque Testing 42
7.3 Stability Testing 43
7.4 Prototype Construction 44
Chapter 8: Completed Design 46
Chapter 9: Conclusion 54
9.1 Proof of Concept 54
9.2 Future Work 54
Works Cited 56
Appendix A: Circuit Schematic 57
Appendix B: Programs 58

List of Figures

Figure 1.1- Diagram of One Box of the Robot 2
Figure 1.2- Illustration of Robot's Positions during One Cycle 3
Figure 1.3- Illustration of Robot Rotating Vertically 4
Figure 2.1- Forward Movement Flowchart 9
Figure 2.2- Rotation Movement Flowchart 10
Figure 3.1- Diagram of forces on the robot showing one set of stability extensions 12
Figure 3.2- One set of stability extensions.....12
Figure 3.3- Rotational Movement configuration 13
Figure 3.4- Side of box with platform 14
Figure 3.5- Side of box without platform attached 14
Figure 5.1- Pololu Wixel Micrcontroller Module 21
Figure 5.2- HS-805BB Hitec Servo 22
Figure 5.3- SHARP Infrared Proximity Sensor 24
Figure 5.4- Output Voltage vs. Distance for IR Sensor 24
Figure 5.5- 298:1 Micro Metal Gearmotor 25
Figure 5.6- TB6612FNG Motor Driver Breakout Board.....26
Figure 5.7- Basic H-Bridge Configuration 27
Figure 5.8- TB6612FNG Inputs and Outputs 28
Figure 5.9- TB6612FNG Clockwise Operating Mode 28
Figure 5.10- MMA7341L accelerometer next to US Quarter.....30
Figure 5.11- Lithium Ion Coin Cell Battery 30
Figure 5.12- LiPo Charger Basic- Mini USB 31
Figure 5.13- LT1528 High Current Voltage Regulator 32
Figure 6.1- Downloading wireless_servocontrol program to the Wixel 33
Figure 6.2- PuTTY Configuration to view output using COM8 at 9600 baud 34
Figure 6.3- Output Compare Modulo Mode 4 35

Figure 7.1- Test Board #1 41
Figure 7.2- Test Board #2 42
Figure 7.3- Testing Output Voltages of the LT1528 Voltage Regulator 44
Figure 8.1- Completed Robot 46
Figure 8.2- Video Snapshots of 180 degree rotation 47
Figure 8.3- Video Snapshots of Obstacle Avoidance 48
Figure 8.4- Completed Circuit Board 49
Figure 8.5- Side of Box with NiMH Power Pack and IR Distance Sensor 50
Figure 8.6- Side of Box with Servo Shaft Connected to the Main Bar 51
Figure 8.7- Side of Box with Power Switch and Charging Switch 52
Figure 8.8- Side of Box with IR Distance Sensor 53
Figure 9.1- Illustration of Robot Climbing Stairs 55

List of Tables

Table 5.1: Component Information 20

ABSTRACT

THE DESIGN OF A SINGLE-LEGGED ROBOT

By

Shari Eskenas

Master of Science in Electrical Engineering

This project involves the design and implementation of a single-legged walking stick robot that utilizes a unique locomotion method. The robot is fully autonomous and can avoid obstacles. Forward locomotion is achieved by rotating the back end of the device up and over the front end. The robot consists of a long bar that connects two boxes that each house electrical and mechanical components. Wireless RF communication is used between the two boxes to coordinate their movements. As a box is lifted, position sensing is provided by an accelerometer and obstacle avoidance is provided by an infrared sensor. A programmable microcontroller manages the control of the components.

Chapter 1. Overview

1.1 Introduction

The most common forms of robotic locomotion include wheeled and multi-legged robots. There are advantages and disadvantages associated with these conventional locomotion methods. Wheeled locomotion has the advantages of good efficiency, balance, and simplicity. However, its disadvantages include difficulties with traction, stability, maneuverability, and control. The robot wheels may not provide enough traction and stability for the desired terrain. Nevertheless, wheeled locomotion remains the most common mobility method in robots and vehicles. Alternatively, the main advantage of multi-legged locomotion consists of adaptability and maneuverability in rough terrain since only a single set of point contacts is required. Additionally, a multi-legged robot can cross a hole or gap as long as its reach exceeds the hole width. The main disadvantages of legged locomotion include high power consumption and mechanical complexity. The leg must be capable of sustaining the robot's weight, and may be required to lift and lower the robot. Also, the legs must have a sufficient number of degrees of freedom in order to achieve high maneuverability.

1.2 Objective

The goal of this project was to create a new method of robotic locomotion whose performance would be comparable to the performance of conventional locomotion systems. The system consists of a rigid bar connected to servo actuators at both extremes. Each servo actuator is in a box that remains in contact with the ground while the robot is at rest. The robot can be envisioned as one rigid leg with a box at each end. As shown in Figure 1.1, each box houses a servo actuator and two motors. The output shafts of the servo and the motor are orthogonal to

each other. The top and bottom of each box are connected to a platform through a motor shaft. The box also contains the electronic components, which include an accelerometer, infrared distance sensors, a microcontroller, a motor controller, and battery supplies.

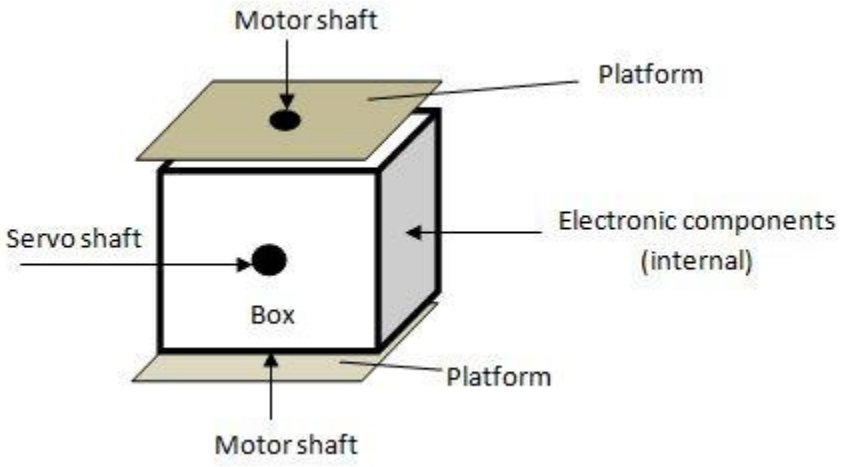


Figure 1.1- Diagram of One Box of the Robot

The mechanism by which the robot moves forward consists of one box that is rotated 180 degrees up and over the other box by the other servo actuator on the opposite end. The servo rotates with variable speed based on the feedback of an accelerometer. An illustration of the robot's locomotion process showing four different positions during the rotation cycle is displayed in Figure 1.2. In step 1, both boxes containing the servos are resting on the ground in the zero degree position. In step 2, the servo contained in the first box starts rotating clockwise and in turn lifts the opposing box off the ground. In step 3, the rotating box has reached the 90 degree position. In step 4, the rotating box enclosure has returned to the ground after having rotated 180 degrees. To continue moving the robot forward, steps one through four are repeated until the robot faces an obstacle.

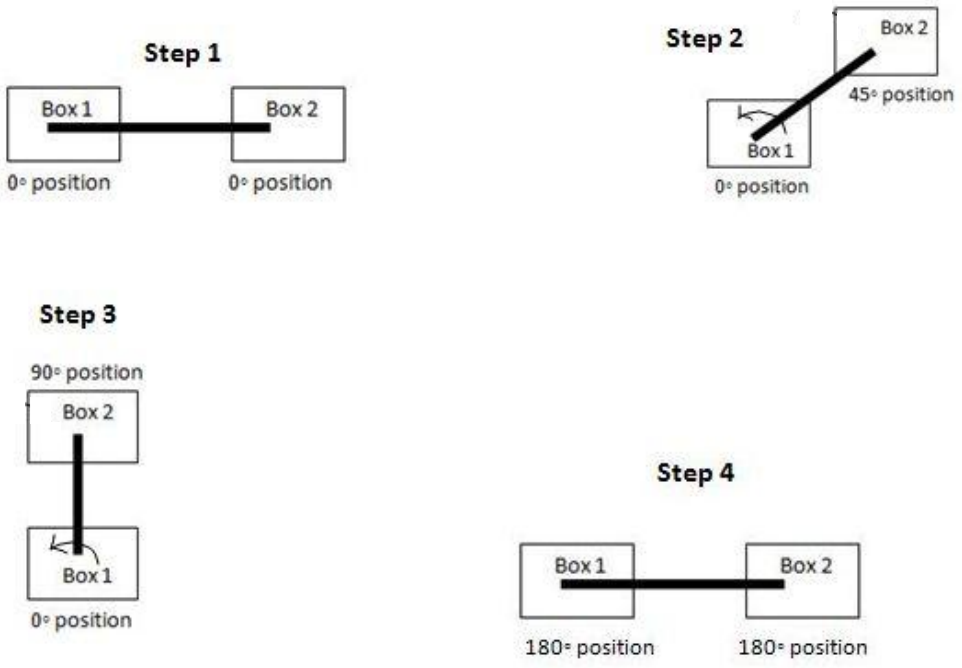


Figure 1.2- Illustration of Robot's Positions during One Cycle

If an obstacle is detected by the infrared distance sensor facing the forward direction before the rotating box is beyond the 90 degree position, an obstacle avoidance routine will be executed. If an obstacle is detected, the rotating servo shaft becomes neutral when the rotating box is at the 90 degree position. The motor attached to the platform that is in contact with the ground then rotates the robot around the vertical axis. An illustration of the robot's configuration when it rotates vertically is shown in Figure 1.3. The motor rotates the robot vertically until the infrared sensor no longer detects an obstacle. Once an obstacle is no longer detected, the robot completes the 180 degree rotation in the new direction.

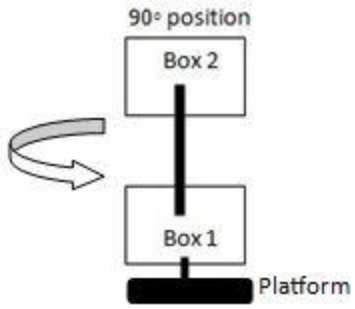


Figure 1.3- Illustration of Robot Rotating Vertically

1.2 Project Outline

In Chapter 2, the system integration of the electrical and mechanical components of the robot is outlined. The algorithm for the robot is also defined.

In Chapter 3, the mechanical design of the robot is discussed. The topics include the overall physical description of the robot, stability mechanism design, rotational axis design, and construction material choices.

In Chapter 4, the hardware design decisions for each component are described. The components include the microcontroller, servo, motors, infrared proximity sensors, motor controller, accelerometer, and power supplies.

In Chapter 5, the component information is provided. This information was retrieved from the datasheets of the components as well as resources listed in Works Cited.

In Chapter 6, the C programming environment and the robot's algorithms are described.

In Chapter 7, electrical and mechanical testing processes are described. The construction of the prototype is also discussed.

In Chapter 8, the completed robot design is presented. Photographs of the final robot are shown as well as snapshots from videos of the robot in motion.

Chapter 9 discusses the conclusion and future work.

The overall circuit schematic is shown in Appendix A. The C programs used in each microcontroller are shown in Appendix B.

Chapter 2. System Integration

This chapter describes the roles of the robot's components within the context of the system. The algorithm for the robot's forward and rotational movements is also described.

2.1 Components

A microcontroller is a device that contains all computer components including the CPU, the memory, the I/O parts, and buses in a single chip. A microcontroller called the Wixel that is programmable in the C language was used in this project to control the operation of the robot. There is a Wixel in each box of the robot. The Wixel has wireless capability, which is used in this project to transmit position sensing data back and forth between each box. The sensors and actuators used in the robot were interfaced with the microcontroller in each box through input/output (I/O) pins. These components include a servo, two motors, an accelerometer, two infrared distance sensors, and a motor controller. Sensor data was read by the microcontroller through analog input pins. The actuators were controlled by the microcontroller's digital output pins, which provided speed control through pulse width modulation (PWM) signals. Pulse width modulation works by changing the signal's average voltage by controlling the amount of time the signal is high during a period.

Each box of the robot is attached to the bar by a servo shaft. A servo contains a motor and control circuitry that provides position feedback and controls the speed of rotation. Servos were selected in this project to provide the necessary torque to allow the robot to lift itself over by 180 degrees. Servos normally do not have the ability to rotate beyond a certain degree due an internal mechanism that provides position feedback. In this project, the servos were specially

ordered to be modified for continuous rotation and consequently position feedback was removed. The speed of the servo could still be controlled by a PWM signal.

Motors were used in this project to connect the top and bottom of each box to a platform that could rotate the robot around the vertical axis to avoid an obstacle. The decision to use motors instead of servos was reached after considering that additional servos were too large to fit within the limited space in each box of the robot. Furthermore, the torque required to rotate the robot around the vertical axis was not as high as the torque required to lift the robot over 180 degrees. Therefore, small motors with less torque capability than the servos were chosen for the purpose of rotating the robot around the vertical axis. The motors were interfaced to a dual motor driver module that was used to control their operation. The motor driver has digital outputs that turn each motor on and off. It also contains a PWM input that is used to provide a varied voltage to the motor inputs.

An accelerometer was used in each box to indicate the angular position of the box. The Wixel microcontroller controls the movement of the robot by utilizing the values of the accelerometer it is connected to as well as the accelerometer values that are wirelessly transmitted to it.

Infrared distance sensors are used to detect and avoid obstacles. Two infrared distance sensors are placed on the robot so that there is always an infrared distance sensor facing the forward direction of movement. Infrared distance sensors were chosen due to their low cost and ability to detect an obstacle within the range of 10-80 cm.

2.2 The Algorithm

An algorithm was created to achieve the robot's forward and rotational movement processes. Accelerometer values are constantly being transmitted and received between each box. There are four possible combinations of final box positions including one box being at 0 degrees and the other box at 180 degrees (and vice versa), both boxes being at 180 degrees, and both boxes being at 0 degrees. This is illustrated in Figure 1.2. The robot is programmed so that one servo is activated based on the box position combination and it will rotate the other box 180 degrees. Once the other box has rotated 180 degrees, the servo turns off and the other servo rotates and repeats the process. This procedure repeats indefinitely until an infrared sensor detects an obstruction. If the IR sensor detects an obstacle and the robot hasn't been rotated past 90 degrees, the servo will hold its position once the rotating box is at 90 degrees. The motor that is attached to the platform contacting the ground will then rotate the entire robot around the vertical axis until the infrared sensor value no longer indicates an obstacle. The motor then turns off and the servo completes the rotation of the box to 180 degrees. Figures 2.1 and 2.2 below show flowcharts of the robot's forward movement and rotational processes, respectively. Each component and sub-process was tested separately before everything was integrated into a final program that governed the robot's behavior.

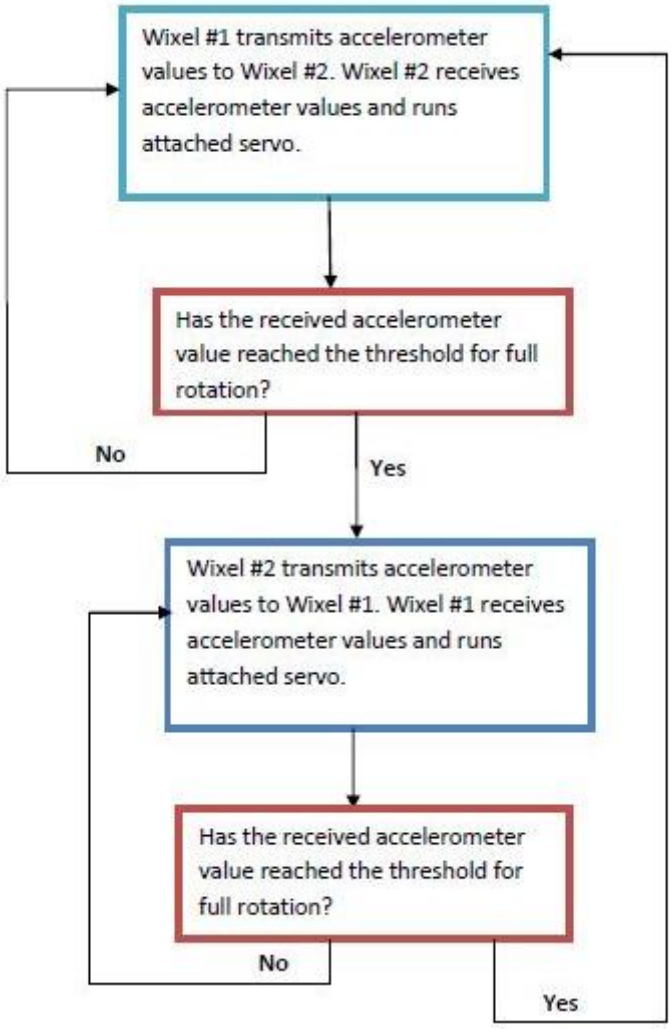


Figure 2.1- Forward Movement Flowchart

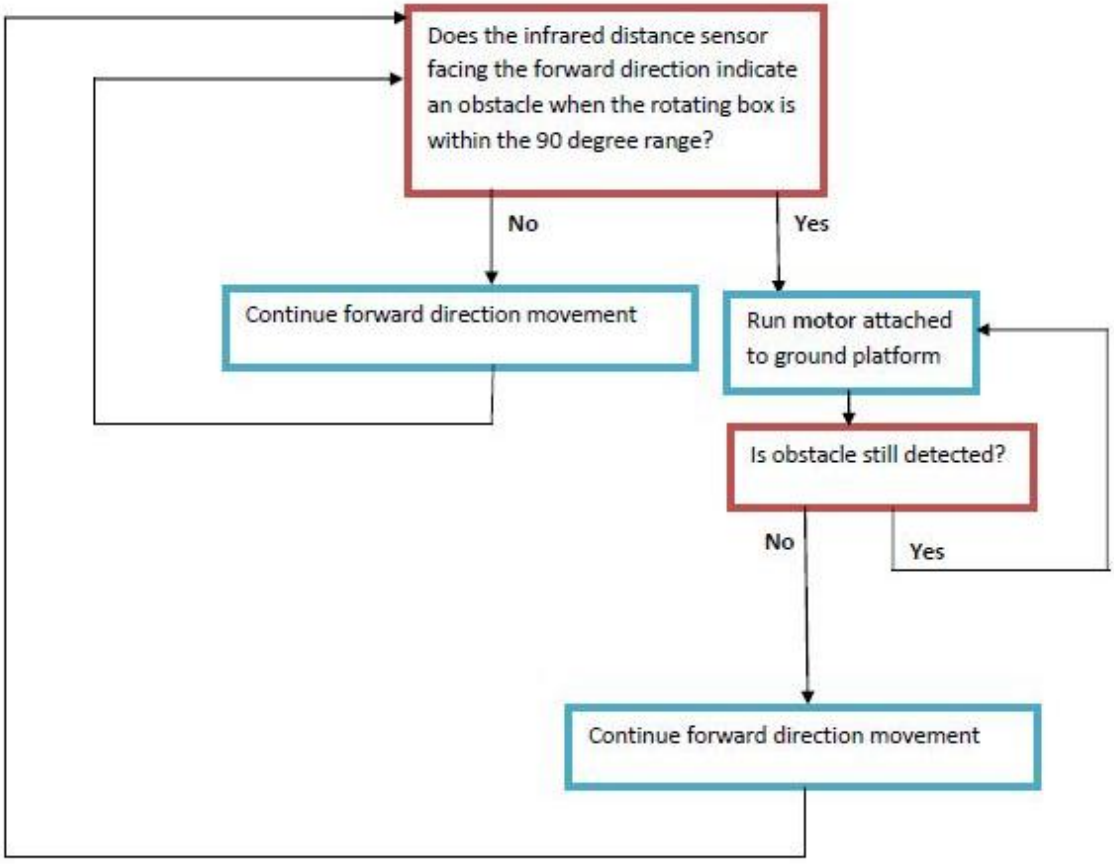


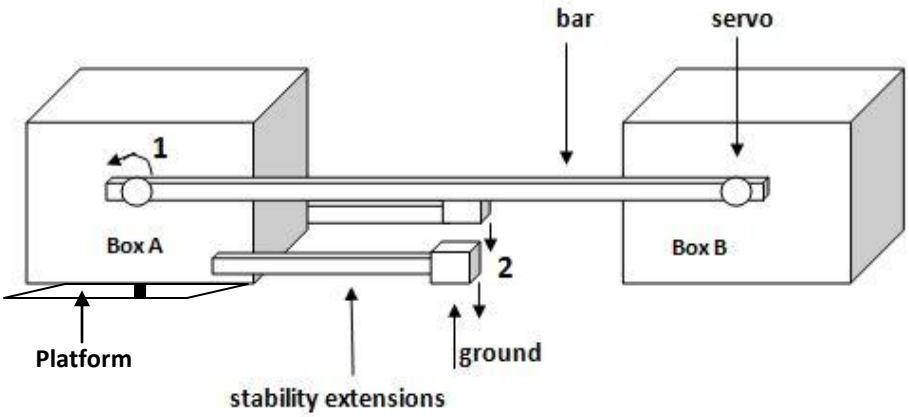
Figure 2.2- Rotation Movement Flowchart

Chapter 3. Mechanical Design

In this chapter, the mechanical design of the robot is discussed. The topics include the overall physical description of the robot, stability mechanism design, rotational axis design, and construction material choices.

3.1 Designing for Stability

The general physical design of the robot is shown in Figure 3.1. The servo contained in "Box A" rotates the other box 180 degrees. The robot's center of gravity is offset from the middle of the robot, which would cause it to tip over during rotation. Therefore, stability extensions are attached to the box to avoid the robot tipping over during rotation. As long as the center of gravity is located between the two extensions, the robot won't tip over. Stability extensions and a platform are shown on only one side of a box for clarity. If the stability extensions were attached to a platform, then during a rotation cycle they might rotate to block the path of the bar that connects the boxes. Many different stability extension designs were attempted in testing. The simplest and lightest design was found to be two "sticks" making contact with the ground at both ends of the box, as shown in Figure 3.1. The stability extensions on each side of the robot were configured slightly differently to avoid interference during a rotation cycle. A photo of one stability extension set is shown in Figure 3.2.



1. The servo shaft attached to the stick rotates, thereby lifting Box B.
2. Box A is restrained from downward motion by stability extension contact with the ground

Figure 3.1- Diagram of forces on the robot showing one set of stability extensions

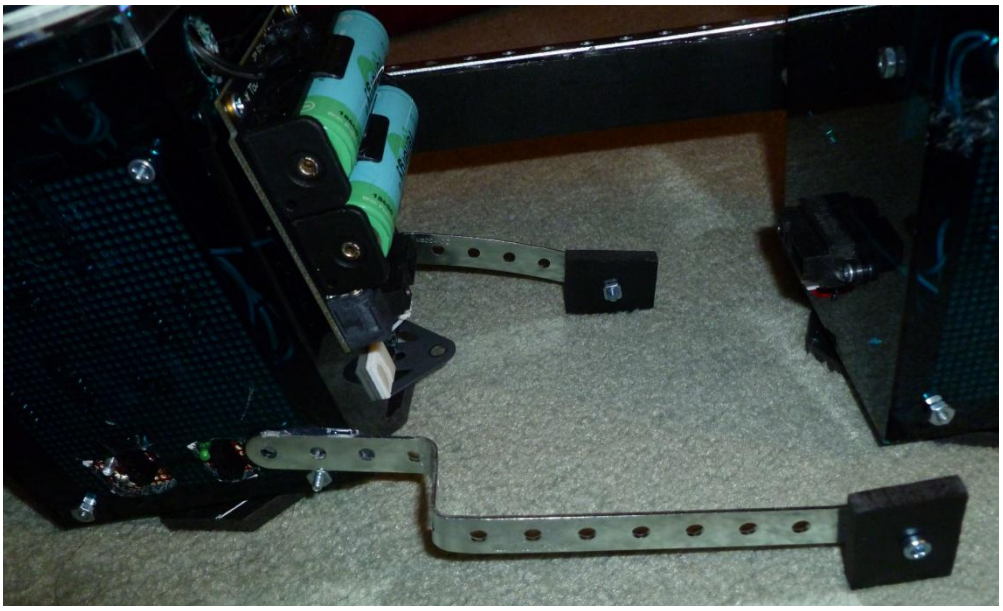


Figure 3.2- One set of stability extensions

3.2 Vertical Axis Rotational Design

A motor shaft protrudes from the top and bottom of each box of the robot to connect to a platform. The robot is able to rotate in a different direction by rotating vertically on the platform that is in contact with the ground. An illustration of the robot's vertical rotation movement configuration is shown in Figure 3.3. Each 6mm thick platform was cut into the shape of a square with the corners cut off to avoid interference with the robot's bar during rotation. A smooth material was needed between the box and the platform to allow it to rotate freely without being constrained by friction. The product "Magic Sliders" from Home Depot worked well beneath each platform to overcome this problem. The Magic Sliders have a very smooth surface intended to easily allow furniture to slide. The Magic Slider squares were attached with an adhesive to each corner of the top and bottom of the box. This solution worked well and the platform rotated freely without getting stuck. Also, the mounting hub that connected the platform with the motor shaft needed to rotate on a surface without too much friction. A nylon washer was placed over the motor shaft beneath the mounting hub so it could rotate on a smooth surface. A side of the box with and without the platform is shown in Figure 3.4 and Figure 3.5, respectively.

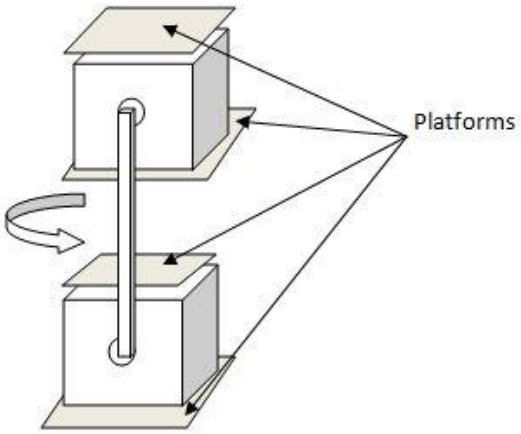


Figure 3.3- Rotational Movement Configuration

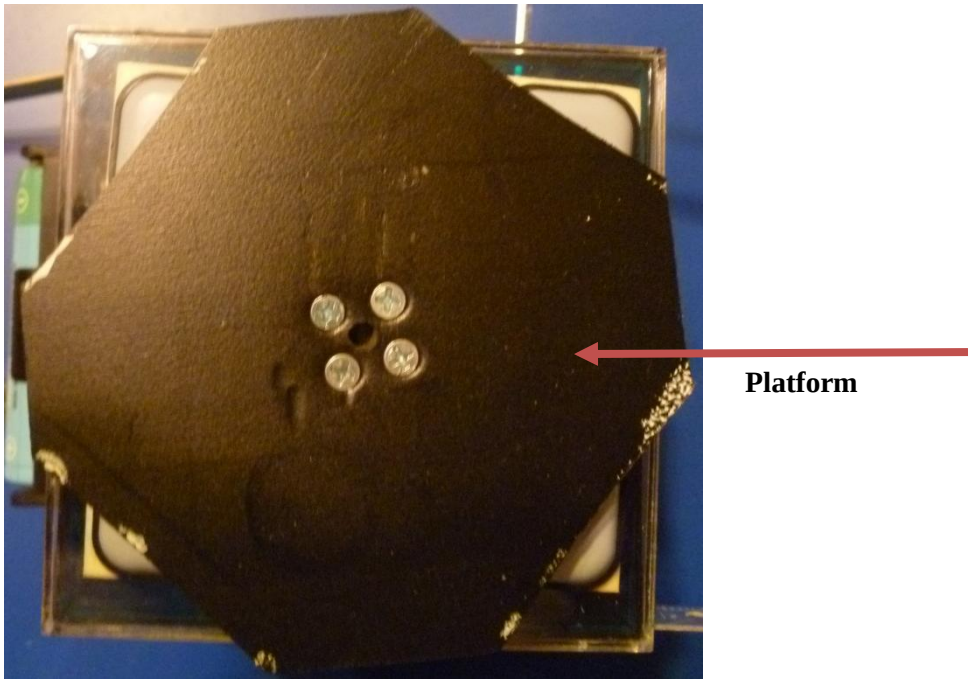


Figure 3.4- Side of box with platform

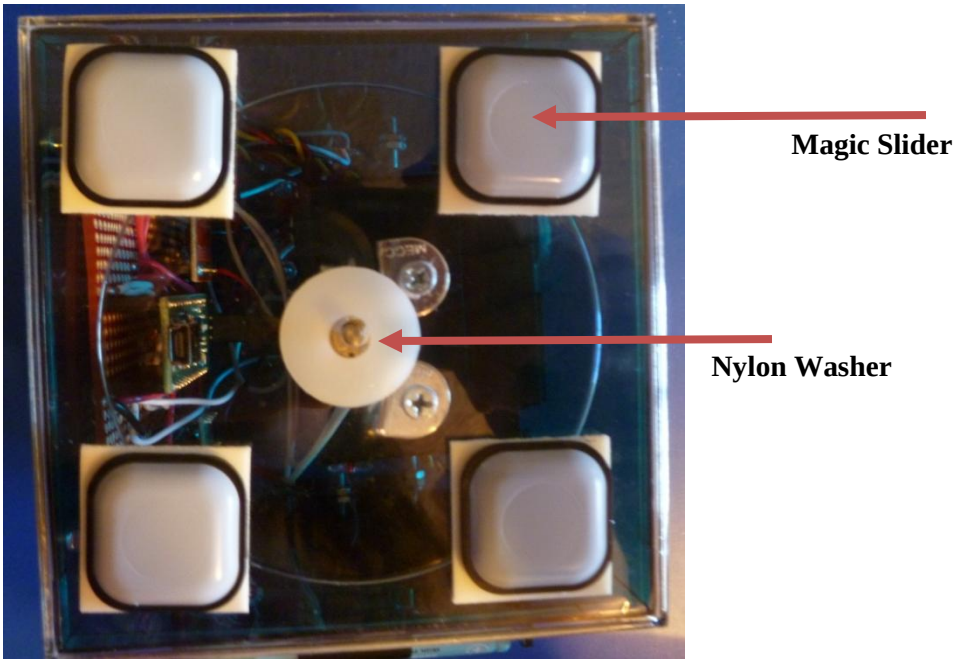


Figure 3.5- Side of box without platform attached

3.3 Construction Material Choices

The plastic material Sintra was used for the boxes' platforms, the bar connecting the boxes, and on the stability extensions. This material was chosen because it is lightweight and easy to cut and drill holes in. The plastic boxes were chosen because they are lightweight and their dimensions were suited to compactly store the electronics. The stability extensions were made from erector set pieces because they can be easily bent and cut into the desired shape. It was important that the stability extensions be easily modifiable because their ideal shape and configuration was found through meticulous experimentation.

Chapter 4. Hardware Design

The hardware design decisions for each component are discussed in this section. The detailed component descriptions are presented in Chapter 5.

4.1 Wixel Microcontroller Module

A small microcontroller module called the Wixel (from Pololu Electronics) was used in this project. It contains a Texas Instruments microcontroller. It was programmed in C in this project. This module was chosen because it has built-in wireless capability. It is the same price as other hobby microcontroller modules that would require a separate wireless module. The Wixel is small, lightweight, and economical ($20). The Wixel was placed at the top of the box below the removable lid for easy access to program it with the USB cable.

The Wixel's pins are configured as analog inputs to read data from the accelerometer and infrared distance sensors. The Wixel's pins are configured as digital outputs to control the operation of the motors. The Wixel also provides the pulse width modulation (PWM) signals to control the speed of the motors and servo.

4.2 HS-805BB Hitec Servo

Digital servos exhibit a higher performance than analog servos, but digital servos are more expensive. The analog HS-805BB Hitec servos were selected for this project because of their lower cost. Each servo occupies the majority of the box space. If a more expensive, smaller servo that supplied sufficient torque were used, the size of each box could be reduced. The servos used in this project are rated for a stall torque of 343 in*oz, which was sufficient for the

torque requirements of the robot. The servo was specially ordered (from SuperDroid Robots) to be modified for continuous rotation. There are many reasons why a continuous rotation servo was chosen instead of a motor. Servos are normally more compact and offer higher torque than motors, which usually require external gearboxes to have comparable torque. In this project, it was important to have an actuator that wasn't too large for the box and also offered significant torque. Also, a servo eliminates the need for an external speed control circuit because a control line is built-in for a PWM input that sets the speed and direction of rotation.

4.3 Sharp GP2Y0A21YK Infrared Proximity Sensors

The infrared distance sensors are used in this project for the purpose of detecting and avoiding obstacles. This allows the robot to be completely autonomous without the need for a remote control to demonstrate its unique method of turning. The SHARP GP2Y0A21YK IR distance sensors were chosen because of their low cost and range of 10-80 cm. The sensors must be able to detect an obstacle at a distance greater than the length of the robot in order to avoid the robot getting stuck once it makes a full forward rotation. The robot measures 53 cm including the stability extensions, which is within the distance sensor's range. Two IR distance sensors are mounted to each box on the two sides that are parallel to the direction of forward motion. This is because one IR distance sensor needs to be facing the front of the robot, which alternates based on the cycle.

4.4 298:1 Micro Metal Gearmotors and Motor Driver 1A Dual TB6612FNG

The gear motors are used to rotate the robot in order to orient it in a different direction of travel. They are attached to platforms that make contact with the ground. The torque requirements of the gear motors did not have to be as high as the torque required of the servos. This is because the gear motors rotate the robot on their axes rather than lifting up a load along a lever arm. Therefore, 298:1 micro metal gearmotors were chosen that have a lower stall torque rating of 40 in*oz. They were also chosen because of their small size, which was required in order for the motors to fit in the box above and below the wide servo. The 1A Dual TB6612FNG motor driver was chosen to control the two motors because of its low cost and ability to drive two motors separately with a PWM signal. It is also convenient that all components are installed on a board and the pins of the TB6612FNG chip are broken out to 0.1" spaced pins.

4.5 MMA7341L 3-Axis Accelerometer ±3/11g

The accelerometer provides analog values that indicate the angular position of the box it is enclosed in. The accelerometer values are used in the C program to control the movements of the robot. The accelerometer used was chosen because of its small size and low cost at only $11.95 from Pololu. It is also conveniently in the form of a breakout board with ten 0.1" spaced pins. It has more than enough functionality, as only one of its three axes is utilized in this project because each box is rotated along one axis.

4.6 Rechargeable Power Supplies and High Current Regulator

A 3.7 V Lithium Ion coin cell rechargeable battery was used as the power source for the Wixel module and accelerometer because of their low current draw. Also, having a separate power supply for the microcontroller allows it to be isolated from inductive noise that can be generated

from the servo and motors. The Wixel has an approximate operating current of up to 30 mA and the accelerometer has a current draw of 0.5 mA. The battery has a current rating of 110 mAh. This coin cell battery was chosen because it is small, lightweight, and rechargeable. Its charger is the LiPo Charger Basic Mini-USB (Sparkfun Electronics), which was placed near the removable lid of the box for easy recharging access.

A 7.2 V

A 7.2 V NiMH rechargeable power pack was chosen as the supply for the servo, motors, and IR sensors. It is a custom-made battery pack purchased from Batteries Plus that was chosen because of its high current capacity and discharge rate that were needed to accommodate the large current draw of the servo. The batteries were also chosen for the power pack because they were lightweight. The battery pack weighs approximately 4 oz. Each battery pack was placed on the outside of each box because of the limited space inside. This power supply was chosen for the IR distance sensor, which typically consumes 30 mA, in order to conserve energy in the coin cell battery.

A high current voltage regulator was needed to regulate the 7.2 V NiMH power pack to 6 V to comply with the voltage ratings of the servo, motors, and IR sensors. The LT1528 from Linear Technology was chosen because of its high current capacity of 3 A. Two external resistors were used with the voltage regulator to obtain the desired 6 V output.

Chapter 5. Component Information

This chapter presents a list of all components and their technical information. The information was retrieved from the components' datasheets in addition to resources listed in Works Cited. A table of all parts, quantities, costs, and suppliers is shown in Table 5.1.

Table 5.1: Component Information

Component	Quantity	Individual Cost	Supplier
HS-805BB Hitec Servo (modified for continuous rotation)	2	$48.40	SuperDroid Robots
Wixel Programmable USB Wireless Module	2	$20.95	Pololu
MMA7341L 3-Axis Accelerometer	2	$11.95	Pololu
Lithium Ion 3.6 V Coin Cell Rechargeable Battery	2	$2.95	Sparkfun
24.5 mm Coin Cell Breakout	2	$2.95	Sparkfun
24.5 mm Coin Cell Holder	2	$0.95	Sparkfun
LiPo Charger Basic- Mini USB	2	$9.95	Sparkfun
Motor Driver 1A Dual TB6612FNG	2	$8.95	Sparkfun
298:1 Micro Metal Gearmotor	4	$15.95	Pololu
Micro Metal Gearmotor Bracket Pair	2	$4.99	Pololu
Universal Aluminum Mounting Hub for 4mm Shaft Pair	2	$6.95	Pololu
Infrared Proximity Sensor- Sharp GP2Y0A21YK	4	$11.95	Pololu
7.2 V NiMH Battery Pack	2	$34.44	Batteries Plus
LT1528 High Current Voltage Regulator	2	$14.26	Sparkfun
SPDT Mini Power Switch	4	$1.50	Sparkfun
ProtoBoard- Wombat	2	$9.95	Sparkfun
Female header: 1x11 pin	2	$0.74	Pololu
Female header: 1x5 pin	4	$0.44	Pololu
Female header: 1x12 pin	2	$0.79	Pololu
Female header: 1x8 pin	4	$0.59	Pololu
Female header: 1x3 pin	2	$0.34	Pololu
Blue LED 3 mm	2	$0.50	Pololu
Green LED 3 mm	2	$0.19	Pololu
Sintra- 8" x 12" x 6 mm	3	$5.75	Solarbotics
Amac Box Turquoise #10022899	2	$3.19	The Container Store
Magic Sliders 15/16 in. Square (8-pack)	2	$6.98	Home Depot
Total cost:		**$505.45**	

5.1 Wixel USB Wireless Microcontroller Module

The Pololu Wixel module contains the CC2511F32 microcontroller from Texas Instruments, which is compatible with the CC2500 transceiver, the CC2510Fx family, and the CC2511Fx family of chips from Texas Instruments. The Wixel has a 2.4 GHz integrated radio transceiver, 32 KB of flash memory, 4 KB of RAM, 2 USARTs (for serial or SPI), 7 timer channels (capable of PWM), and a USB interface. The transceiver consists of a 2.4 GHz PCB trace antenna and RF circuitry. The Wixel has 15 general-purpose I/O lines, including 6 analog inputs connected to a 7-12 bit analog-to-digital converter. The Wixel can be powered from its VIN pin with a 2.7 – 6.5 V source or from its USB port. It is programmable in C through its built-in USB bootloader and the Wixel Software Developer's Kit. A labeled photo of the Wixel is shown in Figure 5.1. It is a small module with dimensions of 0.7" × 1.5".

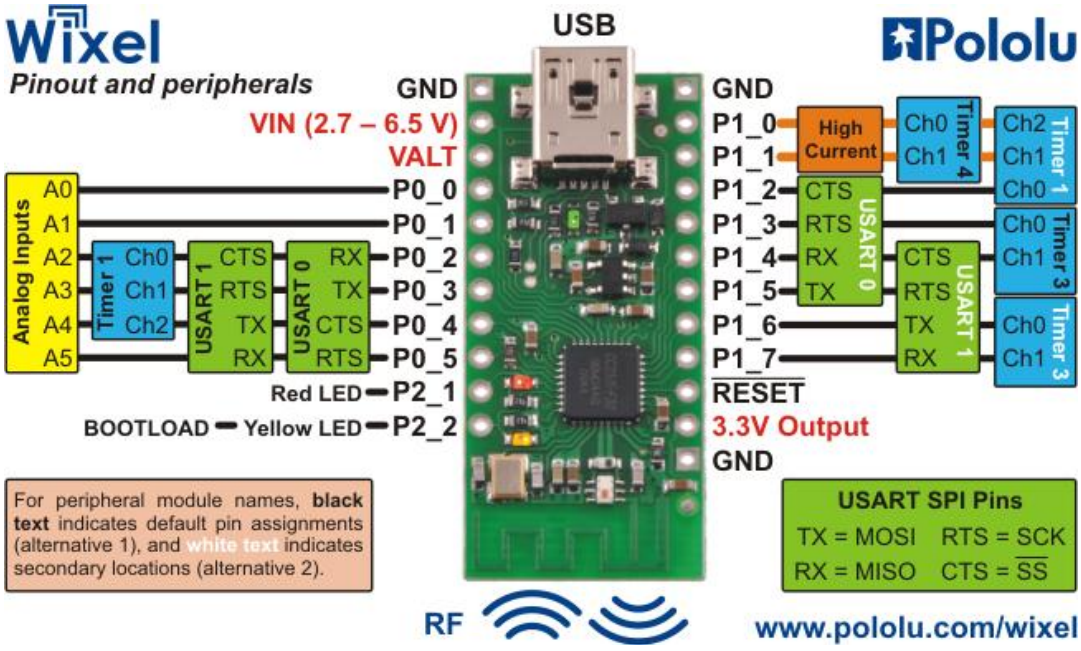


Figure 5.1- Pololu Wixel Microcontroller Module

5.2 HS-805BB Hitec Servo

A servo is different than a dc motor because it uses an external PWM signal to precisely control the position of its shaft. The pulse width of the PWM signal is varied to control the shaft position. The inside of a servo consists of a dc motor, a feedback device, a gearbox, and a control circuit. Externally, there is a drive shaft and three wires for power, ground, and the control signal. The feedback device is usually a potentiometer that has a control dial mechanically linked to the motor. The potentiometer's control dial is rotated with the rotation of the motor shaft. The motor shaft is usually limited to a rotation of 180 degrees because the potentiometer cannot rotate indefinitely. The potentiometer's resistance indicates how far the shaft has been rotated. The control circuit uses this resistance and the PWM input signal to drive the motor to rotate the servo shaft to a certain position and hold.

The HS-805BB is an analog servo rated for 343 in*oz of torque at 6 V. It uses a nylon gear train and a 10mm 15 tooth output shaft. Its dimensions are 2.6"x 1.2"x 2.3". A photo of this servo is shown in Figure 5.2.

Figure 5.2- HS-805BB Hitec Servo

This servo was modified for continuous rotation by SuperDroid Robots. Continuous rotation servos have the advantage over dc motors of containing a gearbox and motor controller. There is a general procedure for modifying a servo for continuous rotation that involves disconnecting the potentiometer from the output shaft. This removes the feedback that normally makes the servo a closed-loop system. Consequently, continuous rotation servos don't have position control capability. The potentiometer (or a resistor) must still be input to the control circuit as a constant resistance. When the servo's PWM input signal commands it to go to a certain angle and the potentiometer feedback is fixed at a different angle, the control circuit will run the motor continually because the feedback will never reach the desired angle. The motor will only stop if the servo is programmed to reach the same angle indicated by the potentiometer value.

5.3 Sharp GP2Y0A21YK Infrared Proximity Sensor

The GP2Y0A21YK0F is an integrated distance measuring sensor unit that consists of a position sensitive detector (PSD), infrared emitting diode (IRED), and a signal processing circuit. It performs triangulation to detect distances. A pulse of light in the wavelength range of 780 nm to 920 nm is emitted and reflected back by an object. The angle at which the light returns is affected by the distance of the reflective object. Triangulation detects this reflective angle to determine distance. The IR distance sensor uses a lens to transmit the reflected light onto an enclosed linear CCD array that determines the angle. The IR distance sensor then outputs a corresponding analog voltage value.

The distance measuring range is 10-80 cm. It provides an analog output voltage that is proportional to the distance. Its supply voltage can be from 4.5 to 5.5 V and its typical

consumption current is 30 mA. A photo of this sensor is shown in Figure 5.3. A graph from the data sheet showing the relationship between output voltage and the distance to the detected object is shown in Figure 5.4.

Figure 5.3- SHARP Infrared Proximity Sensor

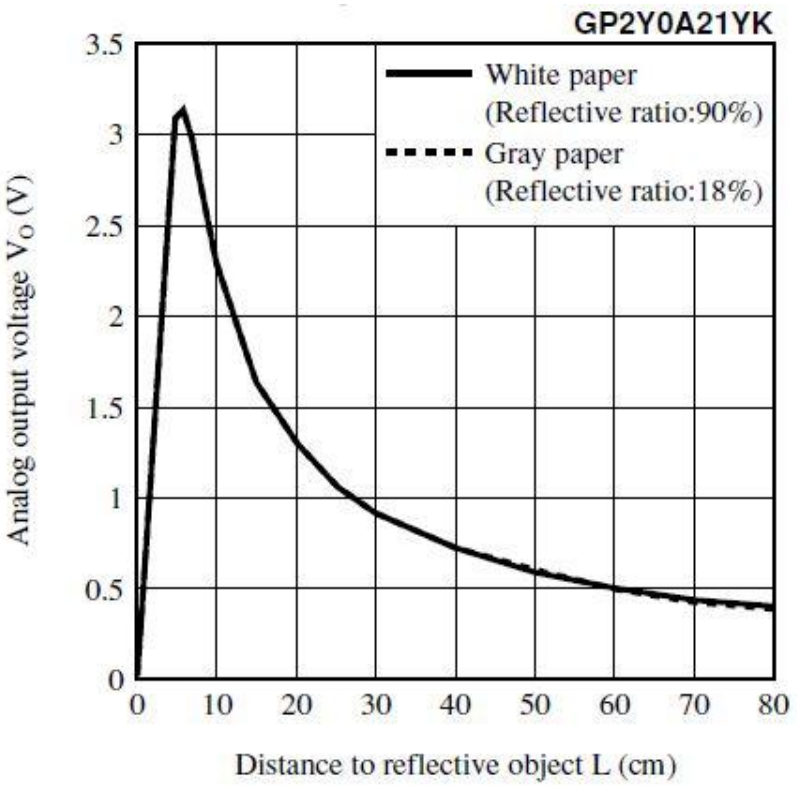


Figure 5.4- Output Voltage vs. Distance for IR Sensor

5.4 298:1 Micro Metal Gearmotor

This brushed DC gearmotor from Pololu contains a 298:1 metal gearbox. Its dimensions are 0.94" x 0.39" x 0.47". At 6 V, the specifications are 45 RPM and 30 mA free-run, 40 oz-in and 0.36 A stall. An image of this gearmotor is shown in Figure 5.5.

Figure 5.5- 298:1 Micro Metal Gearmotor

5.5 Motor Driver 1A Dual TB6612FNG

The TB6612FNG motor driver controls up to two DC motors at a constant current of 1.2 A and peak current of 3.2 A. Two input signals, IN1 and IN2, are used to control the motor in one of four function modes that include clockwise, counterclockwise, short-brake, and stop. The two motor outputs, A and B, are separately controlled and the speed of each motor is controlled with a PWM input signal. An image of the motor driver breakout board offered by Sparkfun Electronics is shown in Figure 5.6.

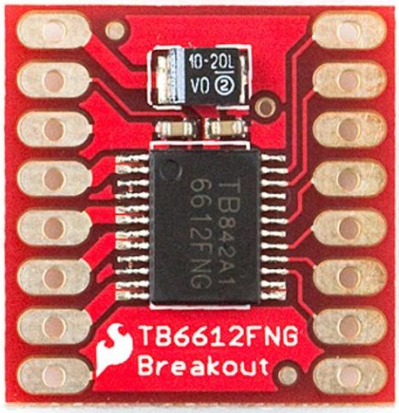


Figure 5.6- TB6612FNG Motor Driver Breakout Board

The PWM signal varies the speed of the motor by varying the average voltage applied to it. The average voltage is set by multiplying the logic high voltage by the duty cycle of the signal, which is the amount of time the signal is high divided by the total period of the signal. An H-bridge circuit controls the motor's direction of rotation by dynamically changing the polarity of the voltage applied to the motor. A basic illustration of an H-bridge circuit that models the four transistors as switches is shown in Figure 5.7.

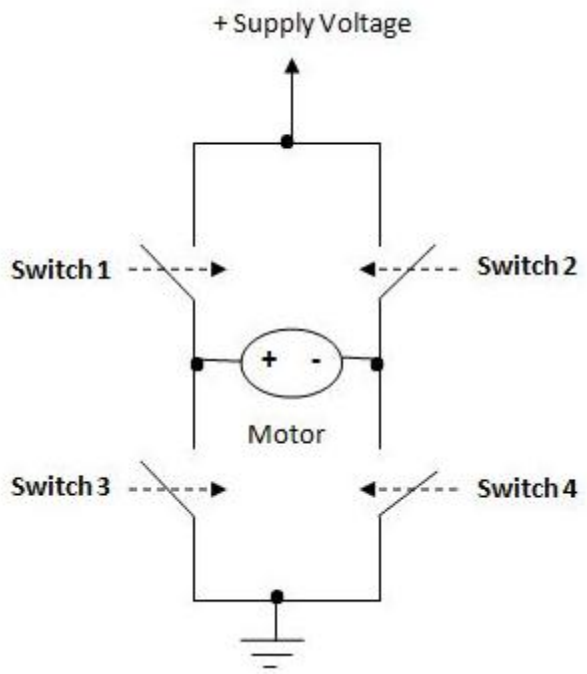


Figure 5.7- Basic H-Bridge Configuration

When switches 1 and 4 are closed, the positive motor terminal is connected to the supply voltage and the negative terminal is connected to Ground, which causes the motor to run clockwise. When switches 2 and 3 are closed, the positive motor terminal is connected to Ground and the negative terminal is connected to the supply voltage, thereby causing the motor to rotate counterclockwise.

The TB6612FNG datasheet provides a table of the function modes resulting from different input values, as shown in Figure 5.8. For this project, the clockwise (CW) function mode was used. The IN2 inputs were directly connected to Ground, the same PWM signal was constantly applied to both PWM inputs, and each IN1 input was connected to an I/O pin that was set in order to run the motor. The H-bridge diagram from the datasheet that illustrates the CW function mode is shown in Figure 5.9. The

diagram shows LD MOS (laterally diffused metal oxide semiconductor) transistors and diodes that protect against back EMF from the motor.

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Figure 5.8- TB6612FNG Inputs and Outputs

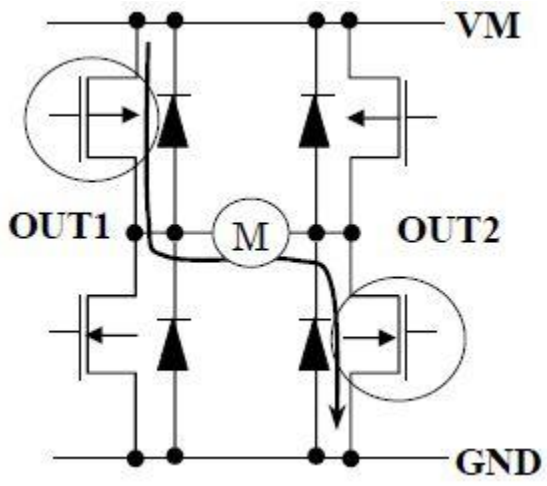


Figure 5.9- TB6612FNG Clockwise Operating Mode

5.6 MMA7341L 3-Axis Accelerometer ±3/11g

The Freescale MMA7341L accelerometer features signal conditioning, a 1-pole low pass filter, temperature compensation, self test, and g-Select that allows for the selection between the sensitivities +/- 3g and +/- 11g. The sensitivity at 3g is 440 mV/g, while the sensitivity at 11g is 117.5 mV/g. The accelerometer has a low current consumption of 0.4 mA and a low voltage operation of 2.2 V-3.6 V. An image of the accelerometer next to a US quarter for size comparison is shown in Figure 5.10.

The accelerometer contains a micromachined capacitive sensing cell (g-cell) and a signal conditioning ASIC. The g-cell can be modeled as a set of movable beams attached to a moveable central mass that moves between fixed beams. The movable beams form two back-to-back capacitors. When the system accelerates, the central mass moves and the distance between the beams changes. This causes each capacitor's value to change because the capacitance is related to distance by $= \frac{A\varepsilon}{D}$, where A is the area of the beam, ε is the dielectric constant, and D is the distance between the beams. The ASIC obtains acceleration data from the difference between the two capacitors. The ASIC also performs signal conditioning and filtering to provide an output voltage that is proportional to the acceleration.

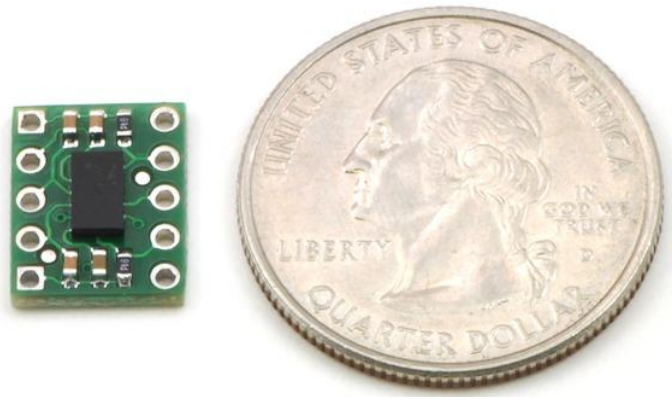


Figure 5.10- MMA7341L accelerometer next to US Quarter

5.7 Coin Cell Rechargeable Battery and LiPo Charger

The Lithium Ion 3.6 V coin cell rechargeable battery is rated for 110 mAh. Its dimensions are 24.5x5.2mm. An image of this battery is shown in Figure 5.11.

Figure 5.11- Lithium Ion Coin Cell Battery

This battery is charged with the LiPo Charger Basic that is available from Sparkfun Electronics. This charger can charge single cell Lithium Ion or Lithium Polymer batteries. A mini USB cable can be connected to it for charging. Its dimensions are 29.4x10.8mm. An image of the charger is shown in Figure 5.12.

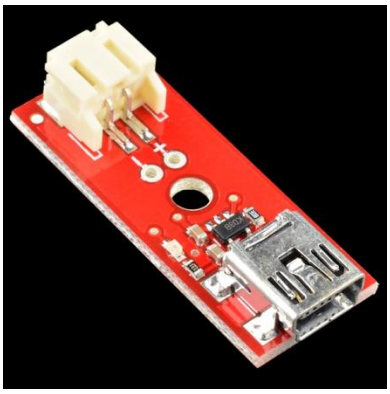


Figure 5.12- LiPo Charger Basic – Mini USB

5.8 NiMH Power Pack

The 7.2 V NiMH Power Pack from Batteries Plus consists of six NiMH batteries that each have a current capacity of 1600 mAh and a discharge rate of 10C. The 10C discharge rate indicates that the battery has the ability to draw a maximum of ten times its capacity. At this current draw of 16,000 mAh, the battery would be discharged in six minutes. This is calculated by first determining $\frac{1600\ mAh}{60\ minutes} = 27\frac{\text{mAh}}{\text{min}}$. Next, this is multiplied by the 10C rating to yield $270\ \frac{\text{mAh}}{\text{min}}$. The battery exhaustion time is then calculated by $\frac{1600\ mAh}{270\ \frac{\text{mAh}}{\text{min}}} = 6$ minutes.

5.9 LT1528 High Current Voltage Regulator

The LT1528 voltage regulator has a high current rating of 3A. The output voltage range can be adjusted from 3.3 V to 14 V using two external resistors. A diagram of the regulator is shown in Figure 5.13. The output voltage is given by the formula $V_{out} = 3.3\ V\left(1 + \frac{R_2}{R_1}\right) + I_{sense}R_{2.}$ An output voltage of 6 V was achieved by using a 220 Ω resistor for R_1 and using the series combination of 150 Ω and 33 Ω resistors for R_2 .

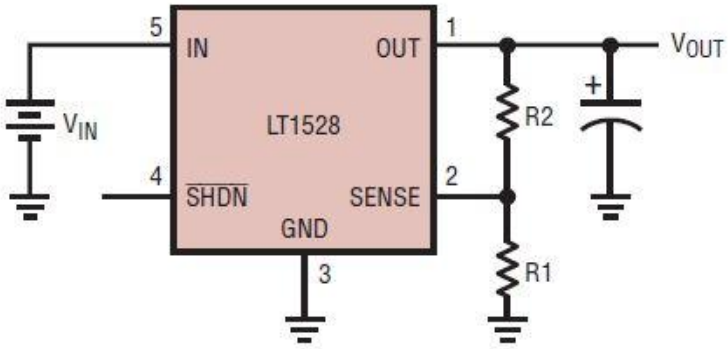


Figure 5.13 – LT1528 High Current Voltage Regulator

Chapter 6. Programming

The final programs for each microcontroller were designed through a bottom-up approach. First, each component of the robot was tested separately in its own program for functionality. Next, programs were written to emulate the behavior of the components in the robot. Once the subprograms worked as intended, they were incorporated into the final program. This chapter discusses the development the C program used in the Wixel microcontroller.

6.1 Development Environment

The free, open source Eclipse IDE was used as the text editor for its advanced C/C++ editing features. Eclipse was set up for use with the Wixel SDK. The Small Device C Compiler (SDCC) was used to compile programs. Programs were compiled and loaded to the Wixel microcontroller using a command prompt, as shown in Figure 6.1.

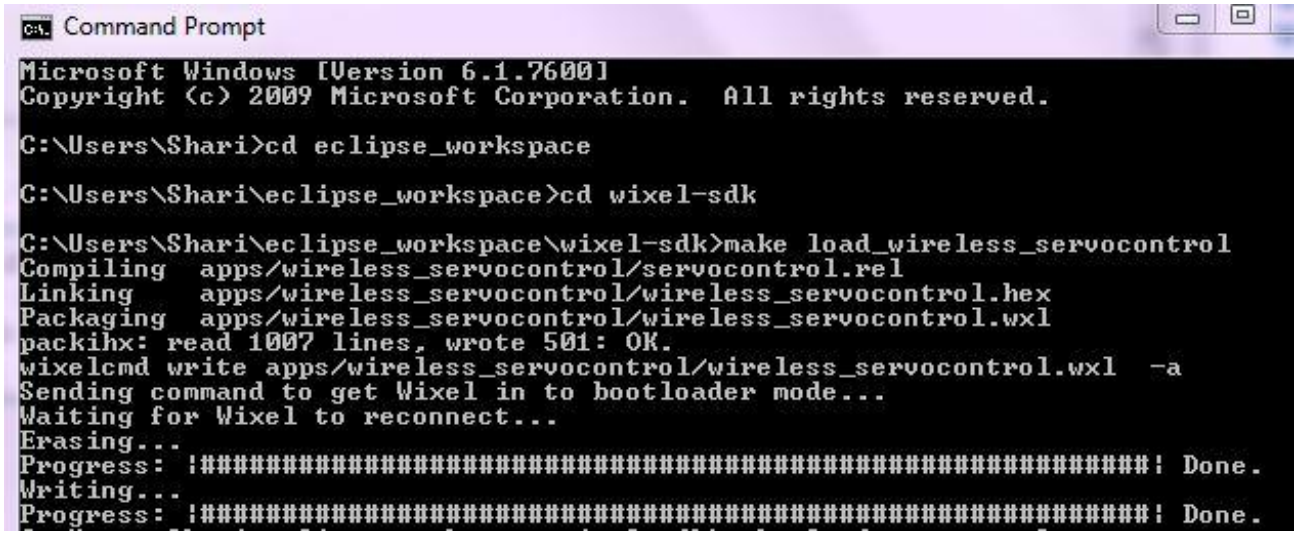


Figure 6.1- Downloading wireless_servocontrol program to the Wixel

The free terminal program PuTTY is capable of sending and receiving bytes on a virtual COM port. It was used in this project to observe sensor data through printf statements. The COM port name and baud rate are specified in the PuTTY window as shown in Figure 6.2.

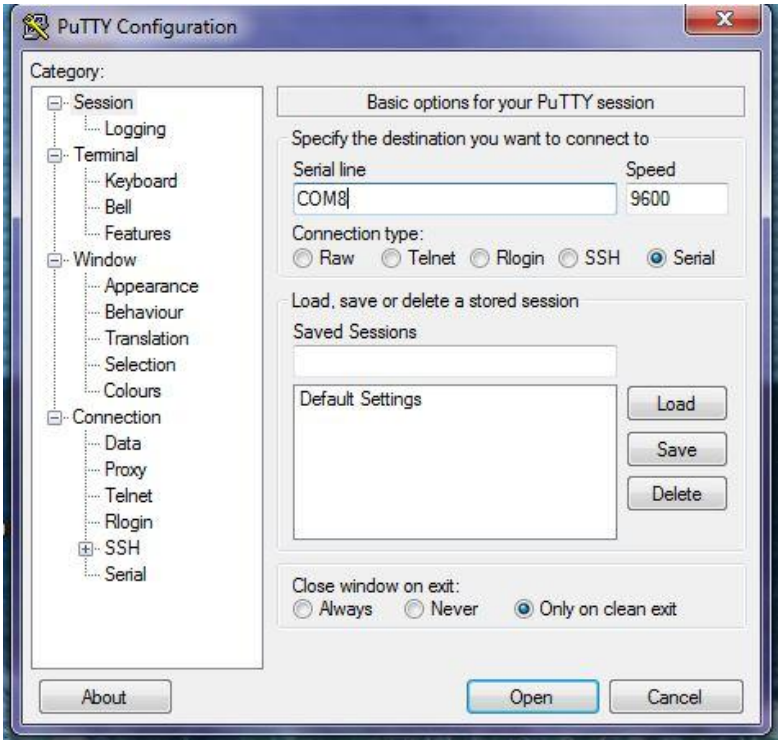


Figure 6.2- PuTTY Configuration to view output in a terminal window using COM8 at 9600 baud

6.2 Pulse Width Modulation

Pulse Width Modulation (PWM) was used to control the speed and direction of the servo and motors in each box. Hardware PWM, which involved manipulating registers, was used rather than manually implementing PWM through software. Hardware PWM is more accurate and consumes less CPU time than software PWM. Unlike hardware PWM, software PWM is not tied to specific pins of the Wixel and could therefore be used to control many servos using a

single timer. However, only two different PWM signals for the servo and motors are required in this project, so hardware PWM was a suitable choice.

The output compare feature of the microcontroller was used to generate the PWM signals for the motors and servo. Timer 1 Channel 1 was used as the PWM source for the servo control line, although Timer 3 Channel 0 had been used for a large duration of the project time. Timer 1 was used instead of Timer 3 because Timer 3 is an 8-bit timer, while Timer 1 is a 16-bit timer. Finer speed control is allowed with a 16-bit timer because of the better pulse width resolution. The Timer 1 Channel 1 Compare Control (T1CCTL1) register settings include compare mode enable, interrupt mask disabled, and an output compare mode of "clear output on compare-up, set on 0". The settings of the Timer 1 control register (T1CTL) include a counter prescaler divider value of 128, disabled interrupts, and a modulo operating mode. The modulo mode causes the counter to repeatedly count from zero to the value stored in register T1CC0. The T1CC0 register stores the 16-bit Timer 1 Channel 0 compare value, which affects the PWM frequency. The T1CC1 register stores a 16-bit compare value that sets the pulse width. The output compare mode method used in this project is illustrated in Figure 6.3.

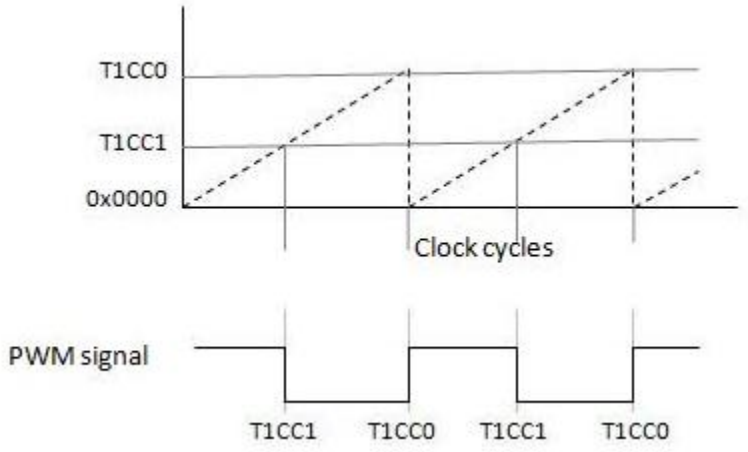


Figure 6.3- Output Compare Modulo Mode 4: Clear Output on Compare-Up, Set on 0

The T1CC1 and T1CC0 register values were calculated for the desired PWM frequency and pulse width. The system timer operates at 24 MHz, and with a prescaler divider of 128, the timer frequency is 0.1875 MHz. For a 50 Hz PWM signal, the T1CC0 value is calculated from the number of cycles as follows:

$$T1CC0 = \frac{0.1875\ MHz}{50\ Hz} = 3750\ cycles = \$0EA6\ (Hex)$$

This indicates that 3,750 cycles of the timer clock are contained in one cycle of the PWM signal. This result is converted into hexadecimal $0EA6 because the registers are programmed in this format. The servo data sheet lists the neutral pulse width as 1.5 ms. To set the pulse width to 1.5 ms, the T1CC1 register value was calculated as follows:

$$T1CC1 = (1.5\ ms)(0.1875\ MHz) \cong 281\ cycles = \$0119$$

It was determined through testing that pulse widths from $0118 to $0120 were approximately neutral. The frequency had to be decreased to below 50 Hz in order for the servo to be completely stopped. To make the servo run clockwise, the T1CC1 value is increased beyond the approximate neutral value. To make the servo run counterclockwise, the T1CC1 value is decreased below the neutral value.

Timer 1 Channel 2 (T1CC2) was used as the PWM source for the two motors. The Timer 1 Channel 2 Compare Control (T1CCTL2) register was configured with the same settings as the T1CCTL1 register used for the servo PWM. The frequency was set by the T1CC0 register in the same manner. The T1CC2 register stores a 16-bit compare value that sets the pulse width. The duty cycle of the PWM signal is defined as the high time of the signal divided by the period of the signal. By increasing the pulse width, the duty cycle increases, and a greater average voltage

is applied to the motor. In this project, a 100% duty cycle was used for the motors to achieve maximum torque. For a 50 Hz PWM signal set by the T1CC0 register, the period is 20 ms and the T1CC2 register value is calculated as follows for 100% duty cycle:

$$T1CC2 = (20\ ms)(0.1875\ MHz) \cong 3{,}750\ cycles = \$0EA6\ (hexadecimal)$$

6.3 Accelerometer Readings

The accelerometer values were significantly fluctuating only while the servo was operating. This was attributed to the vibration of the box due to the force of the servo's movements. Accurate accelerometer values are critical to the correct functioning of the robot. If an accelerometer value is significantly out of range, the program won't execute the correct loop for servo control. To overcome this problem, every 15 accelerometer values were averaged before an average value was transmitted to the other Wixel. It was found through experimentation that averaging 15 values was sufficient to remove inconsistency in the accelerometer readings.

6.4 Forward Movement Logic

The robot moves in the forward direction by default when it is turned on. Accelerometer values are constantly being transmitted back and forth between the two boxes by the receive and transmit function calls in Main. There are four possible position combinations of the two box positions: both Box #1 and Box #2 are at 180 degrees, Box #1 is at 180 degrees and Box #2 is at 0 degrees, Box #2 is at 180 degrees and Box #1 is at 0 degrees, and Box #1 is at 0 degrees and Box #2 is at 0 degrees. Each of these possibilities is tested in the program through an *if…else if* statement. Within each *if* statement, the servo is either turned on or off. The servo control is coordinated between the two boxes so if the servo in one box is turned on, the servo is turned off in the other box. The servos are programmed to only rotate clockwise, so the sequence of box

position combinations is predetermined. Therefore, the *if* statements are used to run the servos in an alternating manner. The servo is turned off by setting the pulse width to a neutral value and lowering the frequency to 20 Hz so less power is applied. The servo isn't turned completely off in order to have it hold its position on the bar as it rotates. The servo is turned on by calling a position control function that controls the speed of the servo. The speed of the servo is dynamically changed based on the accelerometer value received from the rotating box. The highest torque must be applied at the start of the rotation for the servo to be able to lift the box off the ground. The torque must be decreased significantly as the box is lifted to 90 degrees in order to allow the robot to pause at 90 degrees to check if an obstacle is detected. A pulse width of 1.62 ms applied at the beginning of the rotation applies sufficient torque to lift the box. Once the box is at approximately 30 degrees, the pulse width is decreased to 1.54 ms. This causes the box to slowly approach 90 degrees. If no obstacle is detected when the box is within the 90 degree range, the servo pulse width is set to 1.58 ms to complete the 180 degree rotation. Once the box has been rotated 180 degrees, the next *if* statement is executed in each program, and the two servos reverse their on and off roles. This process repeats indefinitely and the robot is able to move forward through repetitive 180 degree rotations.

6.5 Vertical Axis Rotation Logic

The top and bottom of a box each correspond with one accelerometer value because the robot only rotates clockwise in the forward direction. For the box whose servo is running, its own accelerometer value indicates which IR distance sensor is facing the forward direction and which motor is attached to the platform currently on the ground. This allows the program to only take into account the values of the IR distance sensor facing the forward direction to detect obstacles.

It also allows each platform motor to correspond to a particular IR distance sensor so the program will run the appropriate motor based on the forward-facing sensor.

When the received accelerometer value indicates the box is in the 90 degree range, a nested *if* statement tests whether the IR distance sensor facing the forward direction detects an obstacle. If the accelerometer value is below the 90 degree range, it will eventually reach the 90 degree range so it isn't necessary to provide a special command when the IR sensor detects an obstacle. If the accelerometer value is beyond 90 degrees, the 180 degree forward rotation will be completed. This was decided because if rotating box is beyond 90 degrees, it could be in the field of view of the IR distance sensor. If an obstacle is detected by an IR sensor reading that exceeds a threshold, the appropriate motor runs and a neutral pulse width of 1.49 ms is applied to the servo. The whole robot will then rotate vertically until the condition is no longer true and the IR sensor value is below the threshold. The *else* statement will then be executed and the 180 degree rotation will be completed by turning the platform motor off and applying a pulse width of 1.58 ms to the servo. This process allows the robot to resume forward motion in a different direction that is free of obstacles.

Chapter 7. Testing and Construction

This chapter discusses the electrical and mechanical testing involved with the development of the robot. The construction of the prototype following testing is also discussed.

7.1 Electronics and Logic Testing

Before the circuits were soldered, two test boards were created with the circuits on breadboards. Each test board has the configuration of components that would be replicated on each box of the robot. The test boards are shown in Figures 7.1 and 7.2. The test board circuits are identical to those implemented in the actual robot, although there are some hardware differences. One difference is that the test board uses a 3.7 V Lithium Polymer battery as opposed to the NiMH power pack used on the robot. Also, the motors and servo on the test board are cheaper and have less torque capability than those on the robot. This is because the purpose of this test setup was to test the programming logic, and not the torque output of the servo and motors. As described in the Software Design section, each component was separately tested for correct operation with a different program before the final program was tested.

The final goal was to use these boards to simulate the movement of the robot's boxes based on accelerometer and IR distance sensor threshold values. The terminal program PuTTY was used to display the sensor values. Threshold values that would correspond to the box's 0 degree and 180 degree positions were formulated by holding the boards at these positions relative to the accelerometer's x-axis. These threshold values were incorporated in the program to run each servo in an alternating manner based on the accelerometer values. The rotation of the robot in a different direction was also simulated on the test board. A threshold value for the IR distance

sensor was chosen, and each motor was programmed to run based on which IR distance sensor value exceeded the threshold. Each motor corresponds to a motor on the box that is attached to a platform.

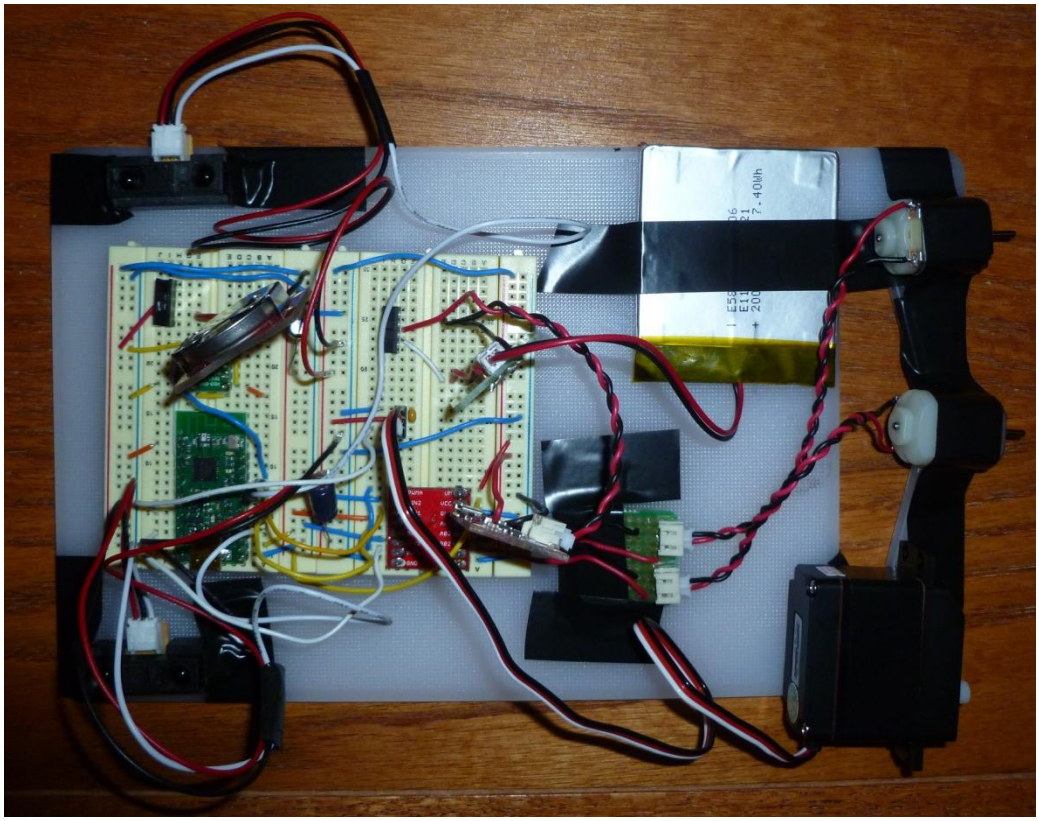


Figure 7.1- Test Board #1

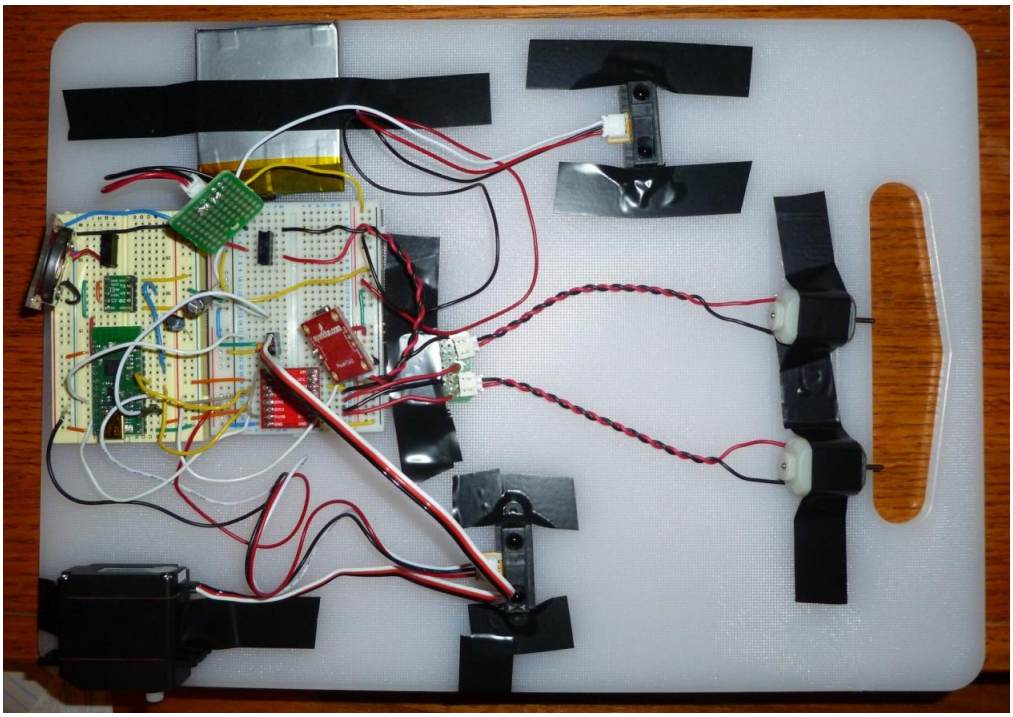


Figure 7.2- Test Board #2

7.2 Torque Testing

The servo is rated by the manufacturer for a stall torque of 343 in*oz, which is the maximum torque that the servo is capable of providing. After attaching a weight to the end of the bar that was ten inches away from the servo shaft, it was found that the servo had only 140 in*oz torque capability. The torque was calculated by multiplying the weight of the load by the distance from the load to the servo. However, after increasing the PWM signal frequency, the servo's torque capability increased and it was able to lift the weight. The PWM frequency affected the servo's torque capability because power is proportional to frequency.

A torque test was performed to determine the stall torque. After attaching more weights to the end of the bar, the stall torque was determined to be approximately 259 in*oz as opposed to 343 in*oz listed in the servo's specification. Subsequently, one of the completed robot boxes was attached to the end of the bar, and the servo was able to rotate the box 180 degrees.

7.3 Stability Testing

A big challenge in this robot design was the issue of stability. The robot is required to lift itself up and not tip over in the process. Testing the robot for stability and modifying the stability extensions was a tedious process. The stability extensions were fine-tuned to fit the dynamics of the robot. The extensions had to be strategically designed and placed in order to avoid interference with other extensions or the box. Each configuration of the four different rotation cycles had to be considered in the stability extension design.

It was an even more difficult task to obtain stability for the robot being held at 90 degrees. As described previously, in order to turn in a different direction, the robot must lift itself up to 90 degrees before it rotates on the platform. The servo cannot be simply turned off once the rotating box reaches 90 degrees because the box can easily fall back down. The torque required to lift the other box off the ground is much higher that the torque required to lift the box at a higher angle such as 45 degrees. If the box rotates too fast, the momentum will cause it to overshoot 90 degrees and fall over. On the other hand, if the servo doesn't supply enough torque, the box will be stalled before it reaches 90 degrees and will fall back down in the direction it lifted from. It was therefore necessary to program the servo to run at different speeds based on the received accelerometer value from the rotating box. This task was difficult because it required both mechanical fine-tuning and meticulous refinement to the servo's PWM frequency and pulse width.

The strategy to rotate the box to 90 degrees and hold it there was to have the servo run with a high torque setting at the beginning of the rotation, and to make the servo run at a low torque setting at about a quarter of the way up so the box would glide to 90 degrees. Ideal pulse width values were determined through testing that prevented the rotating box from overshooting or undershooting the vertical position.

7.4 Prototype Construction

After the final breadboard tests were successful, the circuits were replicated and soldered on a single board. The only additional component that was used on the final board that wasn't on the test board was the LT1528 high current voltage regulator. Resistor values of 220 Ω and 150 Ω in series with 33 Ω that yielded an output voltage of 6 V were determined through experimentation. This test setup is shown in Figure 7.3. This photo shows the voltage regulator soldered to the board and wired to the breadboard. The resistors and NiMH power pack are connected to the breadboard circuit.

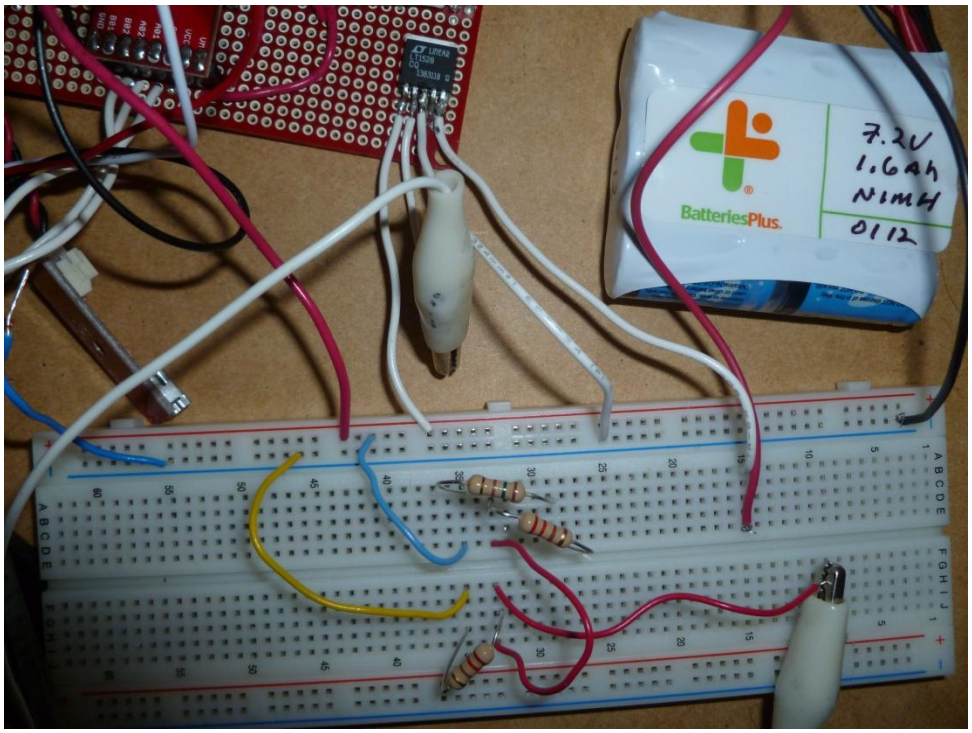


Figure 7.3- Testing Output Voltages of the LT1528 Voltage Regulator

The prototype was designed to be as compact as possible using the best hobby electronics components available at a reasonable cost. The two boxes containing the electronics were assembled in the same manner so both sides of the robot are symmetrical. The robot was constructed by hand and the components were soldered on the boards by hand.

Chapter 8. Completed Design

An overall perspective of the completed design is shown in Figure 8.1. The circuit schematic for the electronics inside of each of the robot's two boxes is shown in Appendix A. The programs used in each Wixel microcontroller module are shown in Appendix B. Three snapshots from a video of the robot moving in the forward direction is shown in Figure 8.2. These images compose a 180 degree rotation. Three video snapshots of the robot detecting an obstacle and rotating away from it are shown in Figure 8.3. The circuit board that is contained in each box is shown in Figure 8.4. The Wixel module, the accelerometer, voltage regulator, coin cell battery, and the motor controller IC can be seen in this photo. The wires protruding from the circuit board are connected to the coin cell battery charger, IR sensors, the power supply, the motors, and the servo.

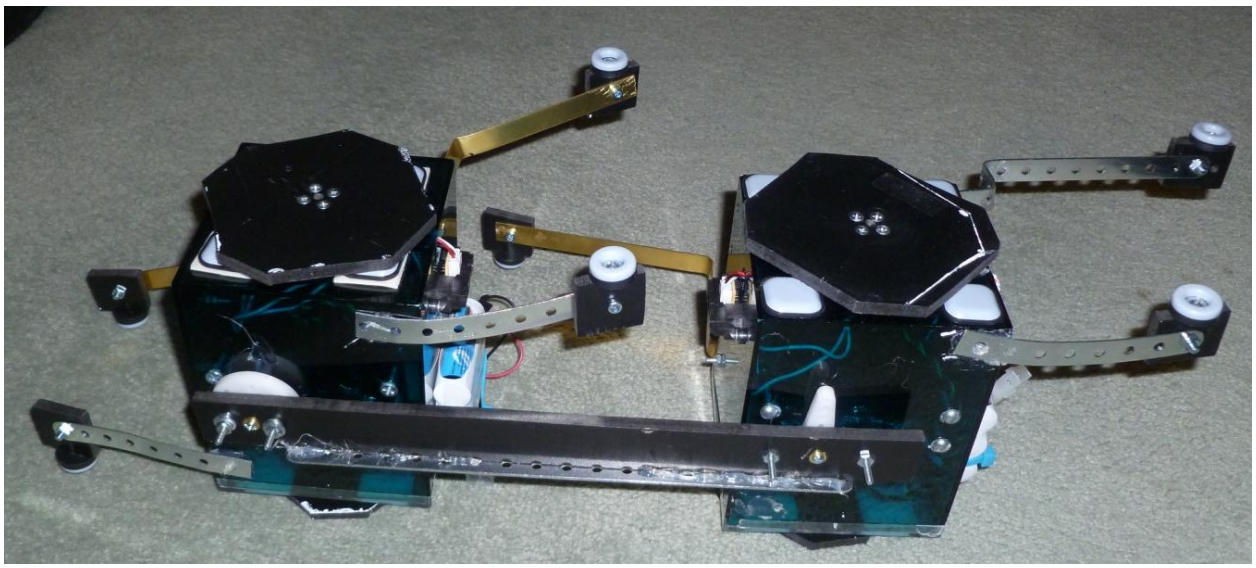
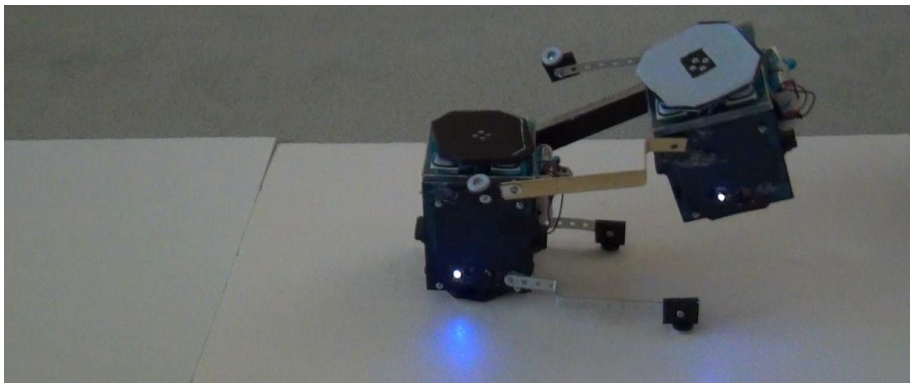
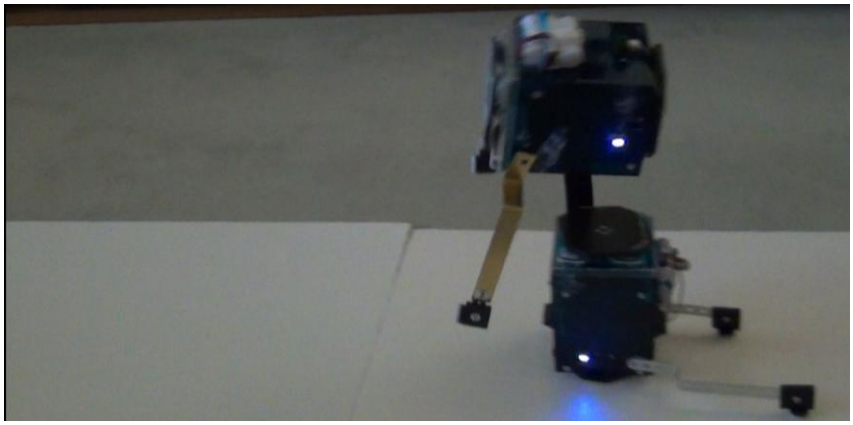


Figure 8.1- Completed Robot

1.



2.



3.

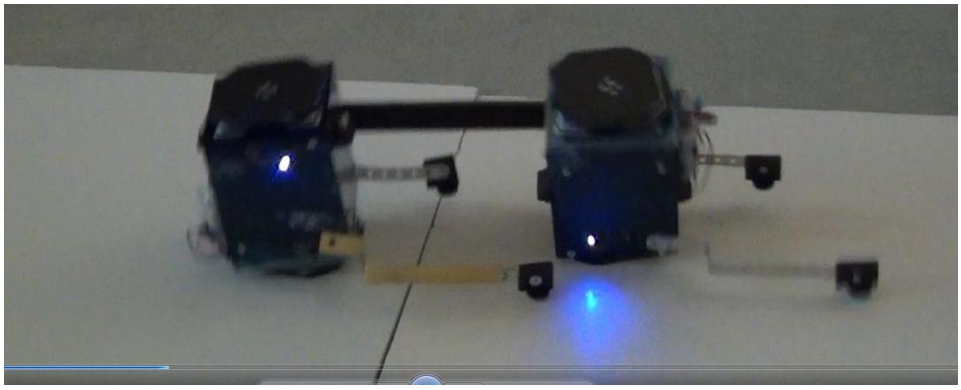
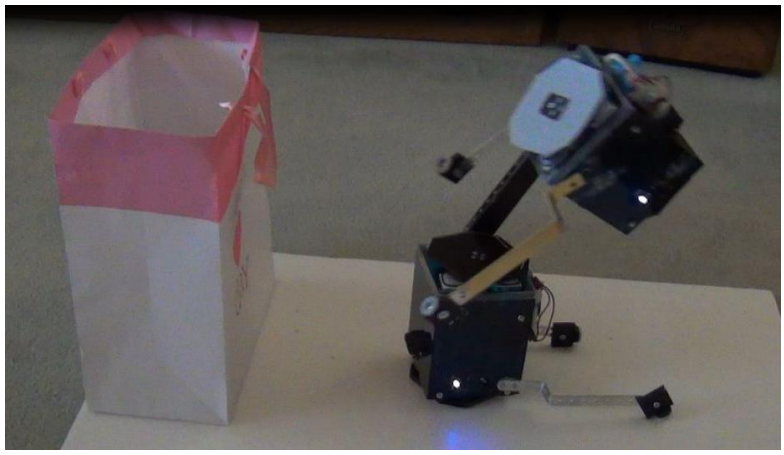


Figure 8.2- Video Snapshots of 180 degree rotation

1.



2.

3.

Figure 8.3- Video Snapshots of Obstacle Avoidance

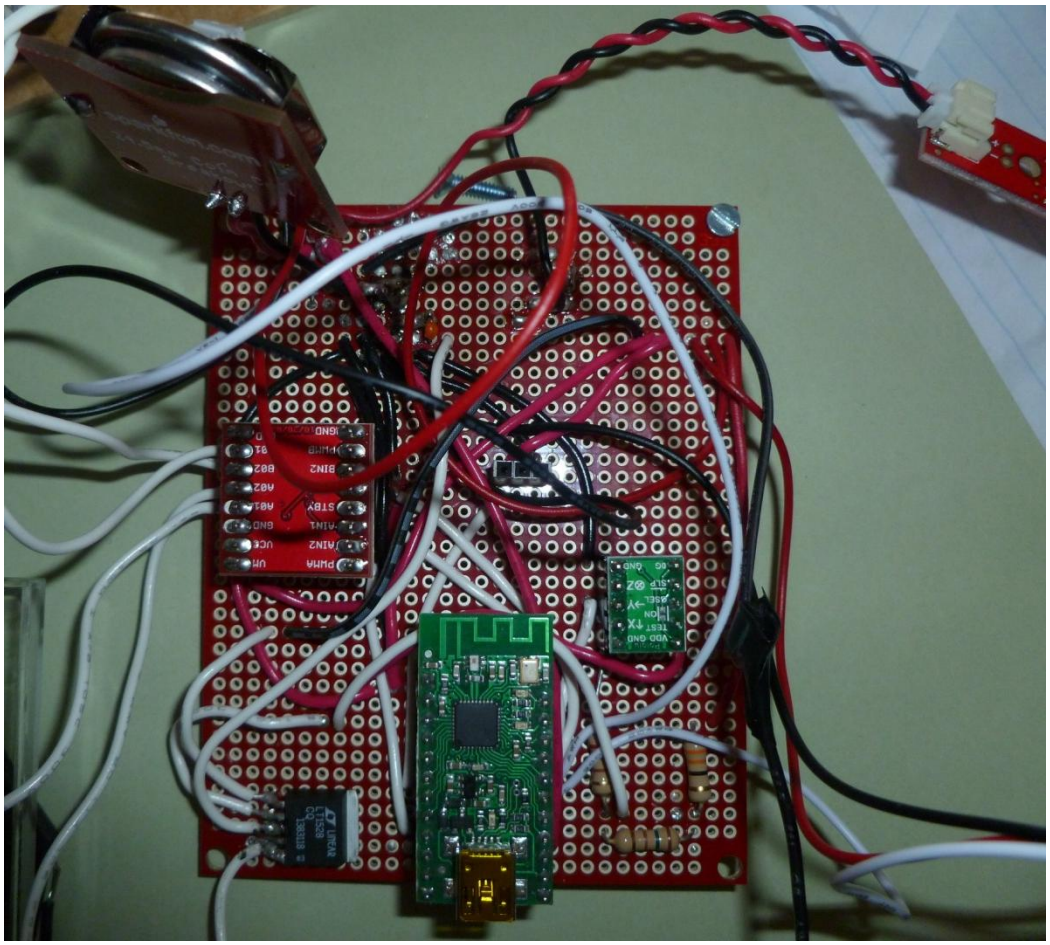


Figure 8.4- Completed Circuit Board

Four sides of a box are shown in Figures 8.5, 8.6, 8.7, and 8.8. The two boxes of the robot are identical in construction and layout.

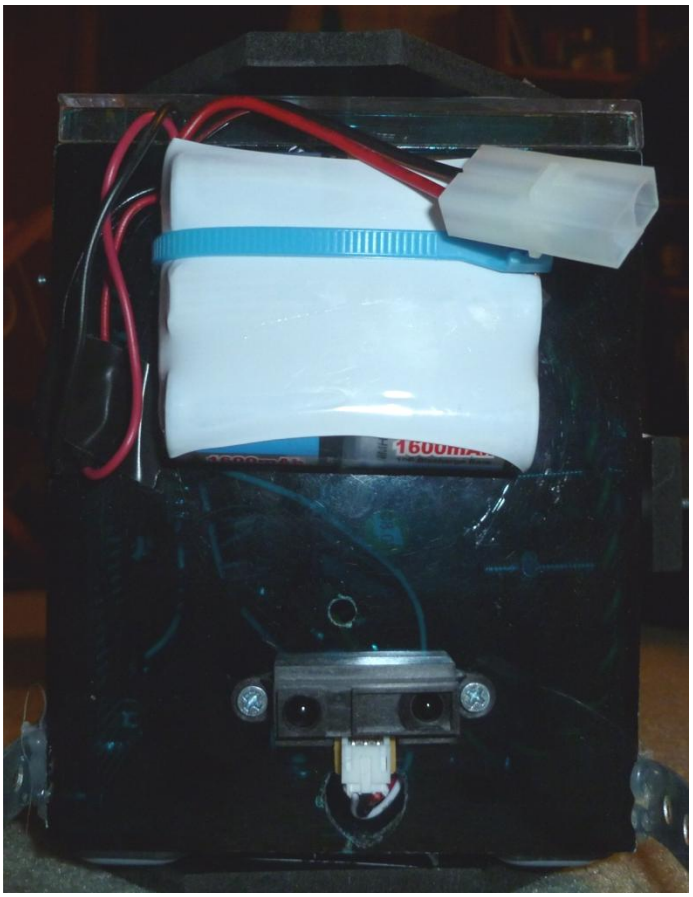


Figure 8.5- Side of Box with NiMH Power Pack and IR Distance Sensor

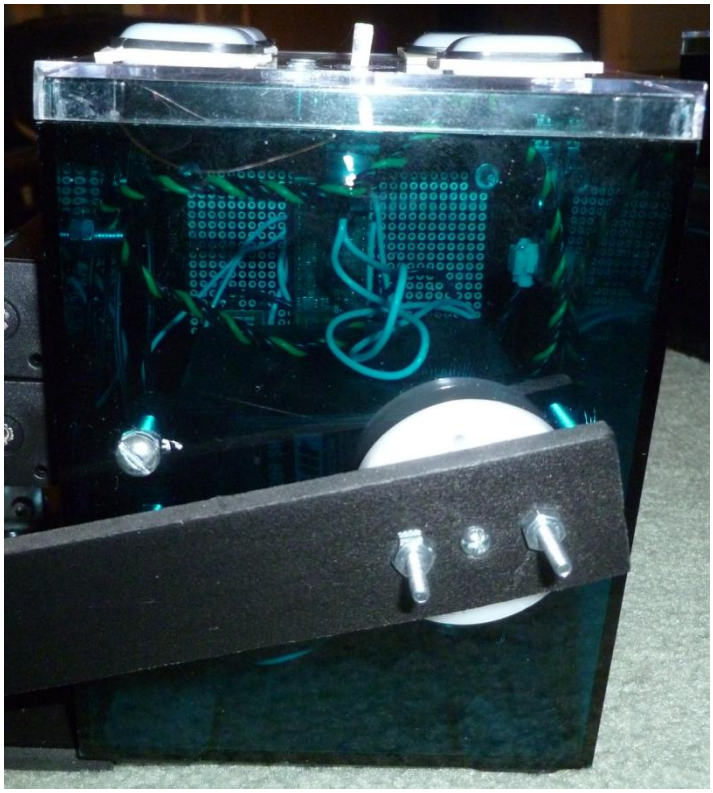


Figure 8.6- Side of Box with Servo Shaft Connected to the Main Bar

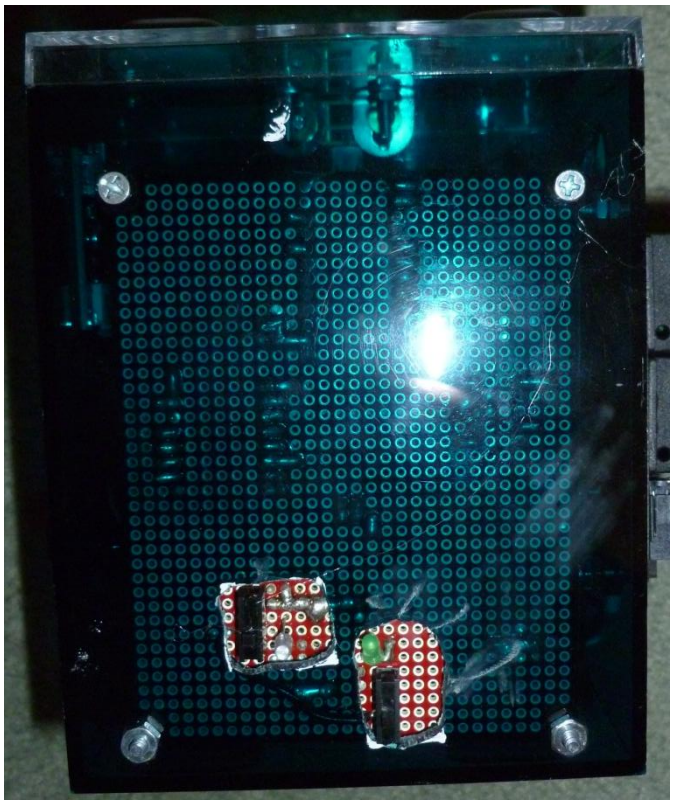


Figure 8.7- Side of Box with Power Switch and Charging Switch

Figure 8.8- Side of Box with IR Distance Sensor

Chapter 9. Conclusion

9.1 Proof of Concept

In this project, a single-legged walking stick robot that utilizes a unique locomotion method has been designed and built. The robot is fully autonomous and can avoid obstacles. The robot was able to perform two cycles of forward rotation with the stability extensions mounted to two sides of the robot. The robot's ability to detect an obstacle and rotate in a different direction was successfully demonstrated. A paper bag was placed in front of the robot, which was detected by the IR sensor facing the forward direction. Once the rotating box reached 90 degrees, the servo rotating the box stopped running and the motor attached to the ground platform rotated the robot vertically. The motor stopped running once the paper bag was out of the field of view of the IR sensor. The 180 degree rotation was then completed by the servo and the rotating box landed in a position that avoided the obstacle.

9.2 Future Work

The robot currently has stability mechanisms on two of the four sides of the boxes that make contact with the ground. This limitation was applied to avoid interference between the stability mechanisms during rotation. In the future, it could be determined how to place stability mechanisms on all sides of the robot without interference. Furthermore, the stability mechanisms could be attached to additional motors that would rotate them out when needed and rotate them inward when they are not in use to avoid interference. This would enable the robot to move forward indefinitely.

The configuration of this robot could be modified to have the capability to traverse stairs. A basic illustration of this possibility is shown in Figure 9.1. The robot currently has a rigid bar

that connects the two boxes on either end. The connection between the two boxes could be modified to allow the robot to climb stairs. One general idea is that the bar could be cut into two parts that are connected with a motor. This motor would rotate to create an angle between the bars. The required electronics would be mounted near the junction of the two bars and could wirelessly communicate with each box.

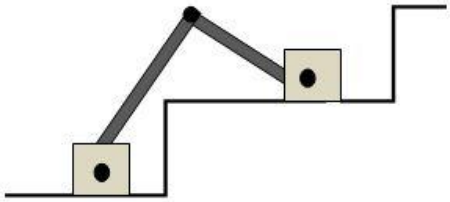


Figure 9.1- Illustration of Robot Climbing Stairs

Works Cited

Freescale Semiconductor. (2008). +/-3g, +/-11g Three Axis Low-g Micromachined Accelerometer. *Freescale Semiconductor Technical Data: Document MMA7341L* .

Malášek, J. (2011, July 26). *Continuous-rotation servos and multi-turn servos* . Retrieved 2012, from Pololu Robotics and Electronics: http://www.pololu.com/blog/24/continuous-rotation-servos-and-multi-turn-servos

Pololu Forum: Reading and Transmitting Accelerometer Values. (2012). Retrieved from Pololu Robotics and Electronics: http://forum.pololu.com/viewtopic.php?f=30&t=4166

Pololu Forum: Wixel and PWM. (2012). Retrieved from Pololu Robotics and Electronics: http://forum.pololu.com/viewtopic.php?f=30&t=4184

Roland Siegward, I. R. (2004). *Introduction to Autonomous Mobile Robots.* Cambridge: The MIT Press.

Scherz, P. (2007). *Practical Electronics for Inventors: 2nd Edition.* New York: The McGraw-Hill Companies.

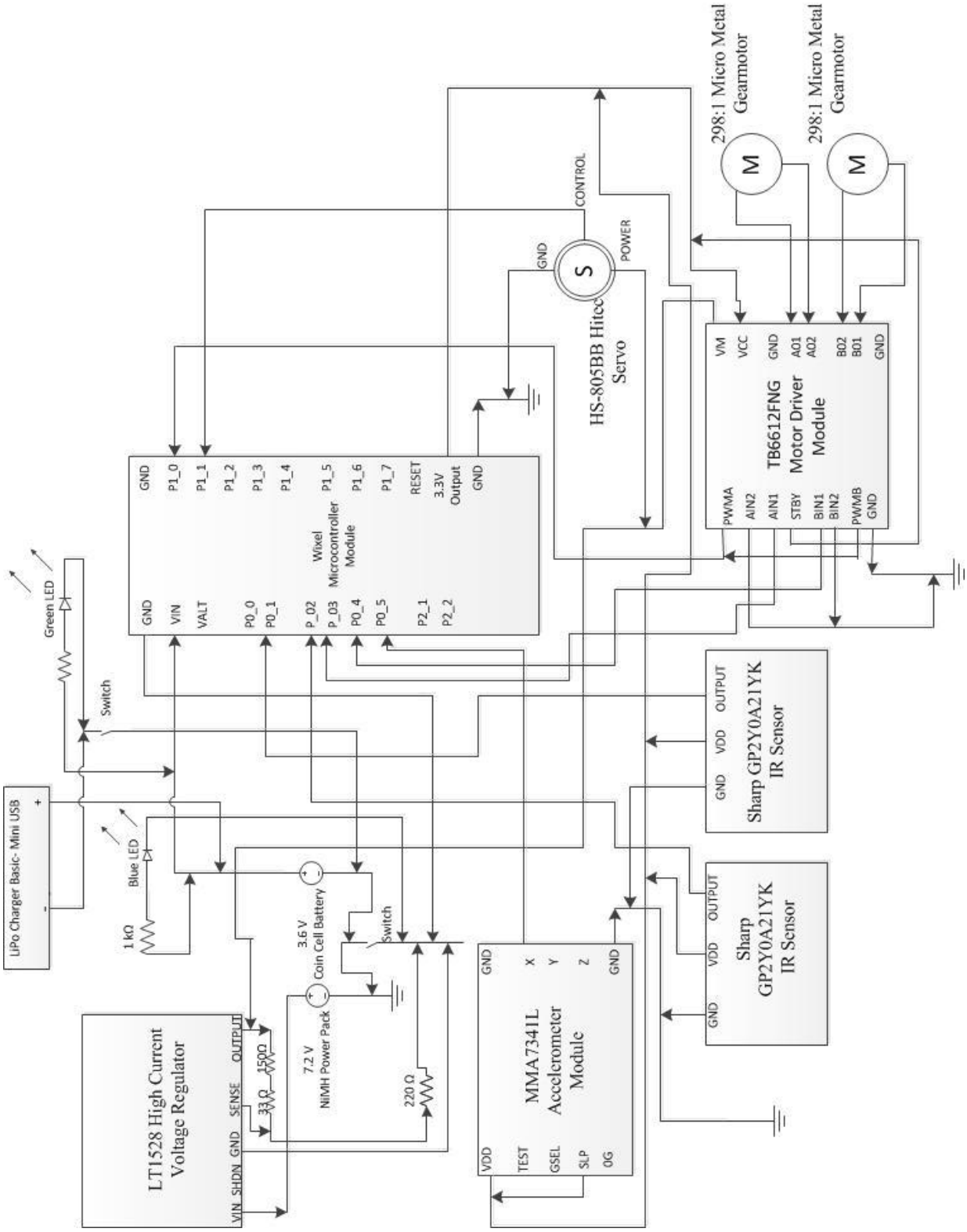
SHARP. (2006, December 01). GP2Y0A21YK0F Distance Measuring Sensor Unit. *Datasheet E4-A00201EN.*

Society of Robots. (2012). Retrieved from SENSORS- SHARP IR RANGE FINDER: http://www.societyofrobots.com/sensors_sharpirrange.shtml

Steven F. Barrett, D. J. (2005). *Embedded Systems: Design and Applications with the 68HC12 and HCS12.* Upper Saddle River: Pearson Prentice Hall.

Toshiba. (n.d.). TB6612FNG Driver IC for Dual DC Motor. *Datasheet* .

Appendix A: Circuit Schematic



Appendix B: Programs

Final Program for Wixel #1

```
/* Filename: FinalDemo1.c
This is the final program used on one Wixel module of the single-legged robot.
The forward motion routine operates normally unless the appropriate IR distance
sensor indicates an obstacle. The IR sensor reading is taken from one of two IR
distance sensors based on the accelerometer value.  If an obstacle is detected, the
appropriate platform motor is run to rotate the robot in a different direction.  This
program is to be run simultaneously with the FinalDemo2.c program on the other Wixel
module.
*/

#include <wixel.h>
#include <usb.h>
#include <usb_com.h>
#include <stdio.h>
#include <radio_queue.h>

/* VARIABLES ************************************************************/

int a_value = 1100;
int my_value = 2100;
PDATA uint16 ir_value1;
PDATA uint16 ir_value2;
int32 CODE param_report_period_ms = 20;
PDATA uint16 loopNumber;
PDATA uint16 ir_sensor;

/* FUNCTIONS ************************************************************/

//initialize PWM for the motors and servo
void PwmInit(){

//sets the frequency to 48 Hz
        T1CC0L = 0x42;
        T1CC0H = 0x0F;

        T1CC2L = 0xA6;
        T1CC2H = 0x0E;

// Timer 1 channel 1 (for servo PWM) set compare mode 4
    T1CCTL1 = 0x24;

// Timer 1 channel 2 (for motor PWM) set compare mode 4
       T1CCTL2 = 0x24;

// Timer 1 set to Alternate 2 location
    PERCFG = 0x40;

// P1_1 set peripheral function which associated with Timer 1 Ch.0 and Ch.1 outputs
```

```
 P1SEL = 0x03;

// set modulo mode prescaler to Tick Freq / 128
    T1CTL = 0x0E;
}


void analogInputsInit()
 {
             //Disable pull-ups and pull-downs for all pins on Port 0.
             P0INP = 0x3F;

}

void adcToRadioService()
{
       static uint16 lastTx = 0;

       uint8 XDATA * txPacket;

       //Check to see if it's time to send a report and if there's a radio TX buffer
available.
       if ((uint16) (getMs() - lastTx) >= param_report_period_ms && (txPacket
                    = radioQueueTxCurrentPacket())) {

             //Both of those conditions are true, so send a report.

             uint8 i;
             uint16 XDATA * ptr = (uint16 XDATA *) &txPacket[5];
             uint16 sum;
             //This should be done before all the ADC readings, which take about
             // 3 ms.
             lastTx = getMs();

             //Byte 0 is the length.
             txPacket[0] = 16;

             //Bytes 1-4 are the serial number.
             txPacket[1] = serialNumber[0];
             txPacket[2] = serialNumber[1];
             txPacket[3] = serialNumber[2];
             txPacket[4] = serialNumber[3];

             adcSetMillivoltCalibration(adcReadVddMillivolts());

                          //average 15 accelerometer values
                          sum = 0;
                          for(i = 0; i < 15; i++){

                          sum = sum + adcConvertToMillivolts(adcRead(5));
                          }
                          my_value = sum/15; //store accelerometer average
```

```
            //Transmit accelerometer average value
            for (i = 0; i < 6; i++) {
                  *(ptr++) = my_value;
            }

            radioQueueTxSendPacket();

      }
}



 void updateLeds()
{
      usbShowStatusWithGreenLed();
}


typedef struct adcReport
{
      uint8 length;
      uint8 serialNumber[4];
      uint16 readings[6];
} adcReport;

void putchar(char c)
{
      usbComTxSendByte(c);
}

void radioToUsbService()
{
      adcReport XDATA * rxPacket;

      //Check if there is a radio packet to report and space in the USB TX buffers
      // to report it.
      if ((rxPacket = (adcReport XDATA *) radioQueueRxCurrentPacket()))
      {
            //We received a packet from a Wixel

            a_value = rxPacket->readings[5]; //store received accelerometer value


            radioQueueRxDoneWithPacket();
      }

}

void ServoService()
{

      T1CC0L = 0x42; //48 Hz frequency
      T1CC0H = 0x0F;

      if (a_value <= 2500 && a_value >= 1920)
```

```
{

	T1CC1L = 0x30; //1.62 ms PWM pulse width
	T1CC1H = 0x01;
}


 if (a_value < 1920 && a_value >= 1800)
 {
	T1CC1L = 0x20; //1.54 ms PWM pulse width
	T1CC1H = 0x01;
 }

if (a_value < 1800)
{
	adcSetMillivoltCalibration(adcReadVddMillivolts());

	if (my_value < 1400) //if this box is at zero degree position
	{
		//test if obstacle is detected by reading appropriate IR sensor
		if (adcConvertToMillivolts(adcRead(1)) > 900 && a_value > 1600)
		{

			setDigitalOutput(3, LOW);
			setDigitalOutput(4, HIGH); //run appropriate motor
			T1CC1L = 0x18; //put servo in neutral
			T1CC1H = 0x01;
		}

		else
		{

			setDigitalOutput(3, LOW); //turn motors off
			setDigitalOutput(4, LOW);
			T1CC1L = 0x28;	//complete 180 degree rotation
			T1CC1H = 0x01;	//with 1.58 ms pulse width
		}

	}


	else //if this box is at 180 degree position
	{
		//test if obstacle is detected by reading appropriate IR sensor
		if (adcConvertToMillivolts(adcRead(2)) > 900 && a_value > 1600)
		{
				setDigitalOutput(3, HIGH); //run appropriate motor
				setDigitalOutput(4, LOW);
				T1CC1L = 0x18;	//put servo in neutral
				T1CC1H = 0x01;
		}

		else
		{
			setDigitalOutput(3, LOW); //turn motors off
```

```
                        setDigitalOutput(4, LOW);
                        T1CC1L = 0x28;      //complete 180 degree rotation
                        T1CC1H = 0x01;      //with 1.58 ms pulse width
                }

            }
        }
}


void PositionControl()
{
       //test if this board is in between 0 and 180 degrees
        if (my_value < 1970 && my_value > 1200 && loopNumber != 4 && loopNumber != 2)
        {
              T1CC0L = 0x9F; //20 Hz frequency
              T1CC0H = 0x24;

              T1CC1L = 0x18; //put servo in neutral
              T1CC1H = 0x01;
        }

       //other board is in between 0 and 180 degrees
       else if (a_value < 1950 && a_value > 1200 && loopNumber != 3 && loopNumber !=
1)
       {

              ServoService(); //call servo control function

       }

       //starting position: both boxes at 0 degrees
       else if (my_value < 1200 && a_value < 1200)
       {

              loopNumber = 1;

              T1CC0L = 0x9F;
              T1CC0H = 0x24;

              T1CC1L = 0x18;
              T1CC1H = 0x01;
       }

       //starting position: this box is at 180 degrees and other box is at 0 degrees
       else if (my_value > 1970 && a_value < 1200)
       {
              loopNumber = 2;

               ServoService();
       }
```

```
	//starting position: both boxes are at 180 degrees
	else if (my_value > 1970 && a_value > 1950)
	{

		loopNumber = 3;

		T1CC0L = 0x9F;
		T1CC0H = 0x24;

		T1CC1L = 0x18;
		T1CC1H = 0x01;
	}

	//starting position: this box is at 0 degrees and other box is at 180 degrees
	else if (my_value < 1200 && a_value > 1950)
	{

		loopNumber = 4;

		ServoService();
	}

}


void main()
{
	systemInit();
	usbInit();
	PwmInit();
	analogInputsInit();
	radioQueueInit();

	while (1)
	{
		boardService();
		updateLeds();
		usbComService();

		radioToUsbService(); //receive accelerometer values
		adcToRadioService(); //transmit accelerometer values

		PositionControl(); //control operation of servos
	}
}
```

Final Program for Wixel #2

```
/* Filename: FinalDemo2.c
This is the final program used on one Wixel module of the single-legged robot.
The forward motion routine operates normally unless the appropriate IR distance
sensor indicates an obstacle. The IR sensor reading is taken from one of two IR
distance sensors based on the accelerometer value.  If an obstacle is detected, the
appropriate platform motor is run to rotate the robot in a different direction.  This
program is to be run simultaneously with the FinalDemo1.c program on the other Wixel
module.
*/

#include <wixel.h>
#include <usb.h>
#include <usb_com.h>
#include <stdio.h>
#include <radio_queue.h>

/* VARIABLES ***********************************************************************/

int a_value = 1100;
int my_value = 2100;
PDATA uint16 ir_value1;
PDATA uint16 ir_value2;
int32 CODE param_report_period_ms = 20;
PDATA uint16 loopNumber;
PDATA uint16 ir_sensor;

/* FUNCTIONS ***********************************************************************/

//initialize PWM for the motors and servo
void PwmInit(){

//sets the frequency to 48 Hz
        T1CC0L = 0x42;
        T1CC0H = 0x0F;

        T1CC2L = 0xA6;
        T1CC2H = 0x0E;

// Timer 1 channel 1 (for servo PWM) set compare mode 4
    T1CCTL1 = 0x24;

// Timer 1 channel 2 (for motor PWM) set compare mode 4
       T1CCTL2 = 0x24;

// Timer 1 set to Alternate 2 location
    PERCFG = 0x40;

// P1_1 set peripheral function which associated with Timer 1 Ch.0 and Ch.1 outputs
 P1SEL = 0x03;

// set modulo mode prescaler to Tick Freq / 128
    T1CTL = 0x0E;
```

```
}


void analogInputsInit()
 {
             //Disable pull-ups and pull-downs for all pins on Port 0.
             P0INP = 0x3F;

}

void adcToRadioService()
{
       static uint16 lastTx = 0;

       uint8 XDATA * txPacket;

       //Check to see if it's time to send a report and if there's a radio TX buffer
       // available.
       if ((uint16) (getMs() - lastTx) >= param_report_period_ms && (txPacket
                    = radioQueueTxCurrentPacket())) {

             //Both of those conditions are true, so send a report.

             uint8 i;
             uint16 XDATA * ptr = (uint16 XDATA *) &txPacket[5];
             uint16 sum;
             //This should be done before all the ADC readings, which take about
             // 3 ms.
             lastTx = getMs();

             //Byte 0 is the length.
             txPacket[0] = 16;

             //Bytes 1-4 are the serial number.
             txPacket[1] = serialNumber[0];
             txPacket[2] = serialNumber[1];
             txPacket[3] = serialNumber[2];
             txPacket[4] = serialNumber[3];

             adcSetMillivoltCalibration(adcReadVddMillivolts());

                          //average 15 accelerometer values
                          sum = 0;
                          for(i = 0; i < 15; i++){

                          sum = sum + adcConvertToMillivolts(adcRead(5));
                          }
                          my_value = sum/15; //store accelerometer average



             //Transmit accelerometer average value
             for (i = 0; i < 6; i++) {
                    *(ptr++) = my_value;
             }
```

```
            radioQueueTxSendPacket();

        }
}



 void updateLeds()
{
        usbShowStatusWithGreenLed();
}


typedef struct adcReport
{
        uint8 length;
        uint8 serialNumber[4];
        uint16 readings[6];
} adcReport;

void putchar(char c)
{
        usbComTxSendByte(c);
}

void radioToUsbService()
{
        adcReport XDATA * rxPacket;

        //Check if there is a radio packet to report and space in the USB TX buffers
        // to report it.
        if ((rxPacket = (adcReport XDATA *) radioQueueRxCurrentPacket()))
        {
                //We received a packet from a Wixel

                a_value = rxPacket->readings[5]; //store received accelerometer value


                radioQueueRxDoneWithPacket();
        }

}

void ServoService()
{

        T1CC0L = 0x42; //48 Hz frequency
        T1CC0H = 0x0F;

        if (a_value <= 2500 && a_value >= 1920)
        {

                T1CC1L = 0x30; //1.62 ms PWM pulse width
                T1CC1H = 0x01;
```

```
}


 if (a_value < 1920 && a_value >= 1800)
 {
       T1CC1L = 0x20; //1.54 ms PWM pulse width
       T1CC1H = 0x01;
 }

if (a_value < 1800)
{
       adcSetMillivoltCalibration(adcReadVddMillivolts());

       if (my_value > 1600) //if this box is at 180 degree position
       {
             //test if obstacle is detected by reading appropriate IR sensor
             if (adcConvertToMillivolts(adcRead(2)) > 900 && a_value > 1400)
             {

                   setDigitalOutput(3, HIGH);
                   setDigitalOutput(4, LOW); //run appropriate motor
                   T1CC1L = 0x18; //put servo in neutral
                   T1CC1H = 0x01;
             }

             else
             {

                   setDigitalOutput(3, LOW); //turn motors off
                   setDigitalOutput(4, LOW);
                   T1CC1L = 0x28;      //complete 180 degree rotation
                   T1CC1H = 0x01;      //with 1.58 ms pulse width
             }

       }


       else //if this box is at 0 degree position
       {
             //test if obstacle is detected by reading appropriate IR sensor
             if (adcConvertToMillivolts(adcRead(1)) > 900 && a_value > 1400)
             {
                         setDigitalOutput(3, LOW); //run appropriate motor
                         setDigitalOutput(4, HIGH);
                         T1CC1L = 0x18;      //put servo in neutral
                         T1CC1H = 0x01;
             }

             else
             {
                   setDigitalOutput(3, LOW); //turn motors off
                   setDigitalOutput(4, LOW);
                   T1CC1L = 0x28;      //complete 180 degree rotation
                   T1CC1H = 0x01;      //with 1.58 ms pulse width
          }
```

```
            }
        }
}



void PositionControl()
{
       //this board is in between 0 and 180 degrees
       if (my_value < 1950 && my_value > 1200 && loopNumber != 3 && loopNumber != 1)
       {

              T1CC0L = 0x9F; //20 Hz frequency
              T1CC0H = 0x24;

              T1CC1L = 0x18; //put servo in neutral
              T1CC1H = 0x01;
       }

       //other board is in between 0 and 180 degrees
       else if (a_value < 1970 && a_value > 1200 && loopNumber != 2 && loopNumber !=
4)
       {

              ServoService(); //call servo control function
        }


        else if (my_value < 1200 && a_value < 1200)
        {

              loopNumber = 1;

               ServoService();
        }

       else if (my_value > 1950 && a_value < 1200)
       {
              loopNumber = 2;

               T1CC0L = 0x9F;
               T1CC0H = 0x24;

               T1CC1L = 0x18;
               T1CC1H = 0x01;

        }


       else if (my_value > 1950 && a_value > 1970)
      {

               loopNumber = 3;
```

```
                ServoService();

        }

        else if (my_value < 1200 && a_value > 1970)
        {
                loopNumber = 4;

                T1CC0L = 0x9F;
                T1CC0H = 0x24;

                T1CC1L = 0x18;
                T1CC1H = 0x01;
         }
}


void main()
{
        systemInit();
        usbInit();
        PwmInit();
        analogInputsInit();
        radioQueueInit();

        while (1)
        {
                boardService();
                updateLeds();
                usbComService();

                radioToUsbService(); //receive accelerometer values
                adcToRadioService(); //transmit accelerometer values

                PositionControl(); //control operation of servos
        }
}
```