Bugs, The Walking Robot

Project Report

Prepared for: San Diego State University

Sponsor: San Diego State University

Adam Tuley
Derek Mack II
Dustin Sackett
Jimmy Doan
Yadira Arevalo
Zhepeng Hong

May 15th 2015

Abstract:

This report address the Rabbit Trails team's final work on a walking robot. This document explains how we designed and built a robot to navigate autonomously, avoid obstacles, climb up and down stairs, and reach a predetermined destination.

TABLE OF CONTENTS

INDEX	Page
1. Introduction	4
2.Hardware	6
2.1 Mechanical Design	6
2.2 Sensor Detection and Placement	9
2.3 PCB Interface and Components	11
3.Software	14
3.1. Walking Algorithm	16
3.2. Turning Algorithm	16
3.3. Navigation	17
4.Conclusions and Recommendations	19
References	24
Appendices	24

Ⅰ.PCB Schematic	24
Ⅱ. Sensor Experiment Data	25
Table 1: Compare Sensor Value with and without Capacity	25
Table 2: Sensor Value vs. Distance	28

Introduction

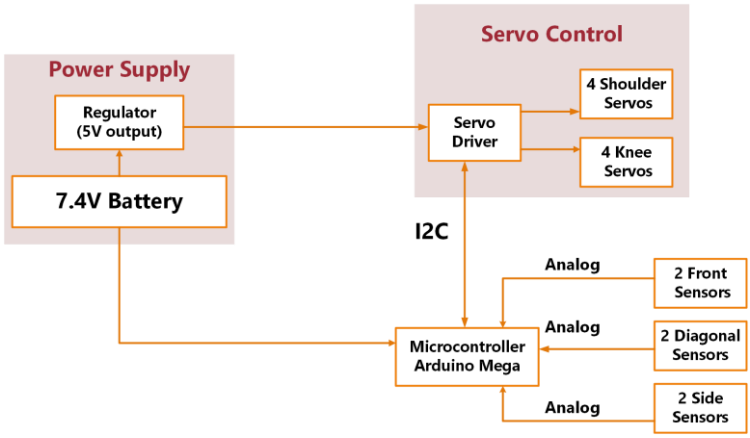
The primary goal of this project was to design and build a walking robot that could perform simple walking tasks. Walking robots are known to have better mobility than wheeled robots but can tip over easily. The robot, therefore, had to adapt to ground conditions in order to walk stably. A design that provided the required foot and torso motion was essential to achieving this goal. There was no remote control allowed during the walking process, meaning the robot did not receive user input at any stage. The robot had the capacity to operate in a closed loop with the help of sensory inputs. It was controlled and actuated using the sensor feedback. There was great consideration for both the mechanical and electronic systems. In this project, the robot was required to fit in a 7 Inch cube.

Specifically, this robot was built to explore a remote location by sensing obstacles and processing the data collected. The robot was able to detect objects in its path by verifying the input received from Infrared Sensors and then able to change its path accordingly.

The robot had a mechanical structure that was comprised of servo motors for the robot's movements. The microcontroller acted as a Central Processing Unit, storing the algorithm that controlled the movement of the robot. With the four footed mechanism used in designing the robot, there were four points of contact with the ground. For optimum balance, the actuators were made to rotate in a sequence and, in turn, the robot would try to balance.

We were faced with two challenges in this project. First, was the fact that the robot had to reach the endpoint within 10 minutes. Taking into account the speed of the robot and the nature of the path it had to follow, this would be difficult to achieve. Also, the dimensions of the stairs were not favorable when compared to the size of the robot. There was a high probability that the robot would tip over while it climbed the stairs.

As the experiment neared its conclusion, the team decided to customize the robot in an attempt to improve its efficiency and accuracy. The idea was to build a custom chassis that would house an onboard power supply and microcontroller. The microcontroller was capable of powering the servo motors and infrared sensors used to identify the position of the robot and help it navigate to the ending point. To the right is a Block Diagram that details the design of our rototic system.



In this paper, we explain the mechanical design and detail how we customised our robot to increase its accuracy. This paper also presents the algorithm we used for walking, turning, and navigating. The procedure used in choosing the components we selected is also explained in detail.

Hardware

2.1 Mechanical Design

When starting this project, the first decision that had to be made was “what kind of robot chassis should we use?”. Given the limited budget of $300 and the requirement to be able to go over stains, our options seemed to be nonexistent. We made the decision that designing the robot ourselves and fabricating it with available CNC machines and 3D printers would be the most cost effective solution. The design was done with Solidworks, and the parts were fabricated with a HAAS TP2 3-axis CNC mill and Makerbot Replicator 2X.

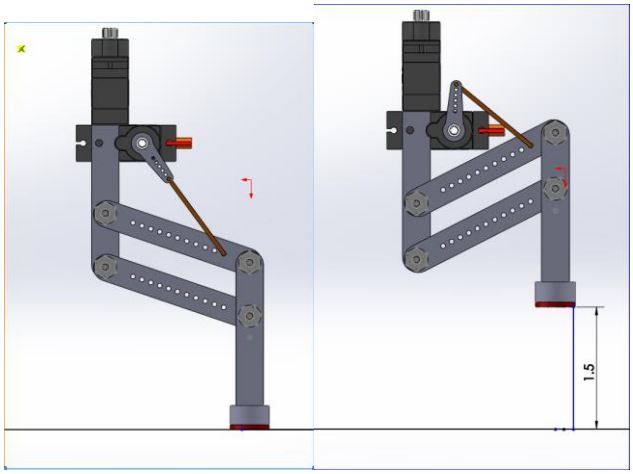
The mechanical design was driven by three main project constraints:

1. The robot must fit in a 7” cube when in its rest position.
2. It must be able to go over 1” high by 2” deep stairs.
3. It must not rely on wheels or any variant of a wheel for locomotion.

Additional Constrains that we designed ourselves:

4. Rigid joints between all moving parts to ensure predictable movements.
5. User friendly assembly and interchangable parts.

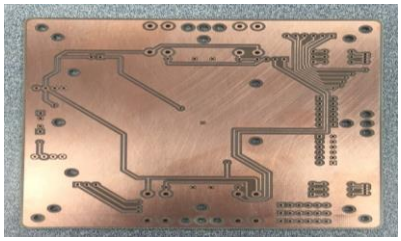
To meet these constraints, it was obvious that we were going to have to use some sort of leg, and said leg would have to have a vertical range of motion of at least one inch while still maintaining a footprint of less than seven inches wide. Below is the first, and final design of our legs (aside from the feet).



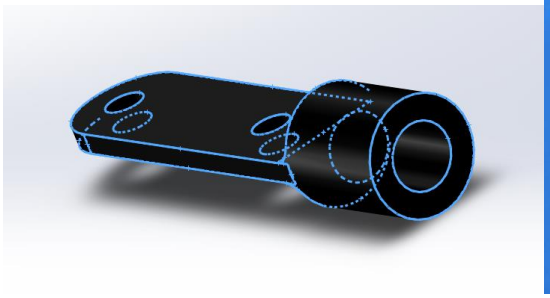
This design allowed for a vertical motion range of one and a half inches.

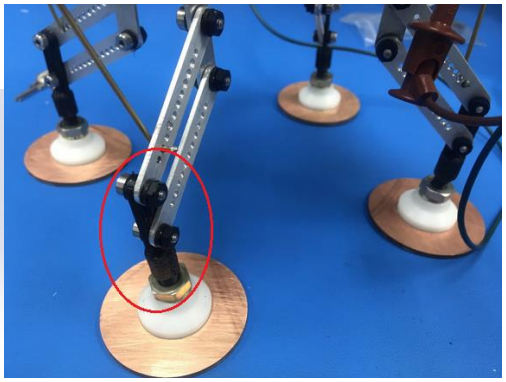
The two servos of each leg were connected to each other using custom servo brackets that we had 3D printed. This part was crucial in ensuring that the legs would move predictably as loose connections between any of the mechanical components proved to be very detrimental to the motion of the legs.

The next aspect of the design to consider was the body that our legs would be connected to. It was decided that we would use a basic square board with many holes in it to allow for various leg placements so we could make a decision for the final body design that would be made out of a PCB. Using a PCB for the body allowed us to be space efficient which was important for our size limit constraint. Below is the final body PCB:



When deciding upon feet, we came to the conclusion that large feet would prove to be beneficial when it came to balancing our robot when it was walking and traversing the stairs. We were not sure exactly as to what dimensions the final feet would be so a lower leg segment was designed that would allow us to swap out the feet quickly and easily. The feet themselves were constructed of ball jointed leveling feet that would give us another degree of freedom- basically acting like an ankle. Below the feet we put on “shoes” that consisted of a circular piece of G10 of various diameters. The final shoes ended up having a diameter of two inches, which provided more than enough surface area to allow the robot to balance on two feet. Below is the lower leg segment along with the feet.





Once all of the parts of our mechanical assembly were fabricated, hardware had to be chosen that would ensure a rigidly assembled robot. To achieve this we used: Precision ground shoulder screws for all of the leg segments, rubber washers that

wouldn’t bind the legs when the nuts were tightened all the way, and split washers on the legs and servos so the nuts would not come off from movement.

2.2 Sensor Detection and Placement

Our robot contained 6 sensors. The model of sensor chosen was the Sharp Infrared Sensor. This sensor was chosen for its effective range of 6 cm to 80 cm. These numbers were verified with our own experimentation. Each sensor was mounted on one of three custom designed brackets. These included one forward facing bracket and two side facing brackets. The front bracket had four sensors mounted on it while each side bracket had only one.

The design of our robot was seven inches tall, while the obstacles defined were one and two inches tall. Instead of placing our sensors on top of the robot and angling them downward, we decided to design the brackets in such a way that they would hang below the robot so that they could point straight out and see the obstacles up to the full 80 cm away. In doing so, we had to ensure that the brackets would not interfere with the stairs while climbing. We were able to resolve this by placing the front facing bracket at ground level with the robot fully crouched. When walking, the robot was in a mostly crouched position and could use the sensors to detect walls and stairs. When climbing the stairs, it would raise itself to its tallest position and thus raise the bracket and clear the stairs.

Two sensors on the front bracket faced directly forward while the other two were mounted facing diagonally outward from the front of the robot. The front two forward facing sensors were offset both vertiacally and horizontally. Due to the vertical offset,

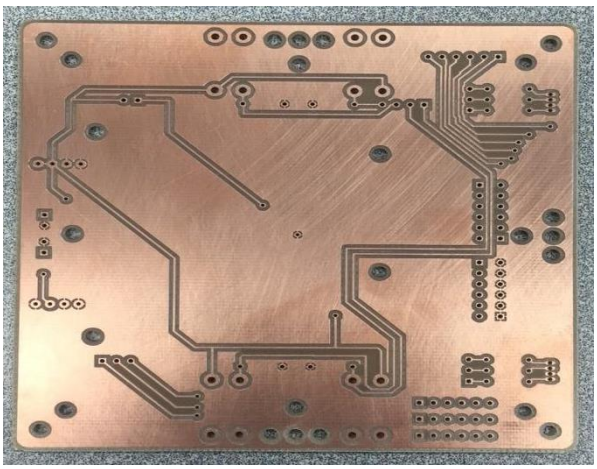
the robot was able to differentiate between stairs and walls. When facing a wall, both vertically offset sensors return the same value. When facing the stairs, the vertically offset sensors would return values approximately 2 inches different from eachother. Due to the horizontal offset, the robot was able to ensure that it was always square with the course. If the robot started walking crookedly, these sensors would return different values and a command would be sent to turn the robot until the issue was resolved.

The other two sensors on the front bracket were used for centering on the stairs. The width of the stairs was the same as the width of the robot. Because of this, we had to ensure that the robot was perfectly centered in front of the stairs before attempting to climb them. These two sensors were mounted diagonally. At a predtermined distance from the front of the stairs, the robot would only see the stairs with these two sensors if it was centered in front of them. The robot was programmed to go to that predetermined distance and not get any closer until both sensors detected the stairs. It would strafe from side to side until this happened. Only when this was accomplished, would the robot attempt to climb the stairs.

The side sensors were used to detect obstacles on the side of the robot. If an obstacle prevented the robot from moving forward, it would be forced to turn and travel side to side. While doing so, the side sensors would look for available forward paths since traveling forward was our primary objective. These side sensors also provided a mechanism for keeping the robot centered when walking down a path. If the two values returned by the side sensors were different, within some limit, the robot would be able to tell that it was not centered. A series of turning algorithms

could then be called to recenter the robot and give it the maximum probability of locating the stairs.

2.3 PCB Interface and Components

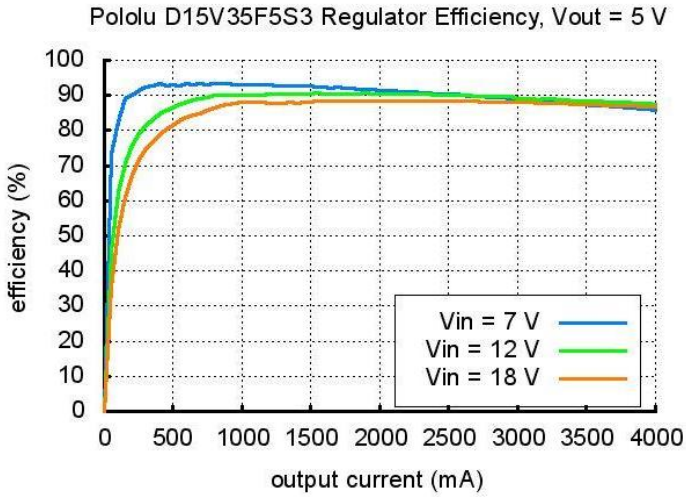


We designed a Printed Circuit Board to interconnect and power each of the components. Initially, we used a blank piece of G10 as our chassis and connected all of our components to the microcontroller and the servo shield. This was used as a prototype to ensure that everything worked as expected. The PCB provided more secure connections and improved the asthetic appeal of our robot. The components that were connected to the board included the battery, micro-controller, voltage regulator, servos, and sensor brackets.

We selected a 7.4 V Lithium Ion battery for our main source of power. We had to ensure that the battery could provide at least 2 A of current and would last a reasonable amount of time for the purpose of software development. This battery provided up to 4 A of current and more than an hour and a half of use for our application. The battery was connected directly to the microcontroller. The effective input voltage of our microcontroller was within that of the battery's output so no special consideration was needed. We attached a second connector to the battery to feed a step down voltage regulator. The output of this regulator would provide a steady 5 V to our servo controller and servo motors. The servo controller was an

ancillary board referred to as a “Sheild.” This simply means that it was a board specifically designed to interface with Arduino microcrollers. The voltage regulator we selected was a Pololu 5 V Step Down Regulator, which provided excellent efficiency at the voltage levels we were using. Below are pictures of the battery and voltage regulator we selected.

From the datasheet shown below, the voltage regulator is about 90% efficient when connected to our 7.4 V battery.



We selected the Adruino-Mega for our microcontroller. This provided sufficient memory and processing power for our robot and an easy platform to code on.

We selected the MG90S Mirco Servos for our motors. These met our requirements for power, size, torque, speed, and price. They also contained metal gears as opposed to plastic, which was a high selling point. These servos have a maximum range of 180 degrees but we decided to limit that range to 140 degrees to decrease the liklihood of damaging them in software development. Below are pictures of the Arduino microcontroller and servos that we selected.

3. Software

Servo Driver Software Incorporation

Although the addition of the Adafruit servo controller diminished the amount of wiring on the robot design, it also affected the coding style needed during our software development. Our microcontroller, the Arduino mega, posseses the ability to utilize useful libraries like the Servo library to keep coding fast and simple. The Servo library allows the user to simply set a degree they would like the motor to reach and the microcontroller outputs the correct PWM signal to the correct pin as shown below.

```
myservo.write(pos);
```

Unfortunately, the servo driver is able to take input only from specific PWM signals and not commands in degrees. Below, we can see the format for this function: the pin on the servo driver connected to the servo to move the beginning of the pulse followed by the end of the pulse.

```
pwm.setPWM(servonum, 0, pulselen);
```

Although not hard to understand, keeping track of which pulse corresponded with which degree was an inconvenience. We also had to be careful to understand which PWM signals would drive the servo motors to their mechanical limits so that these could be avoided. Early in the development process we decided to limit our servos to a maximum range of 140 degrees, from 20 degrees to 160 degrees. We worked around the troublesome servo driver code by matching each PWM signal output when using the Servo library and mapping the rest of the signals in between as degree values as well. Here is an example of setting up a pulse to send our servo motors to degree 60.

```
pulse60 = map(60,0,180,SERVOMIN,SERVOMAX);
```

The SERVOMIN and SERVOMAX variables refer to the minimum and maximum pulse length count out of 4096. After finishing the pulse setup, the team moved on to writing functions for basic movements.

Basic Movements

In order for the robot to be able to complete any course, it first needed to be able to perform basic movement operations which could be called upon when needed. For easier servo identification, they were all named according to their joint position. As described by our mechanical design, there were 2 servos per joint, the servos in the

back were named either a knee or a hip, the ones on the front are named shoulder or elbow.

```
int rk = 3;
int lk = 1;
int rh = 2;
int lh = 0;
int ls = 4;
int le = 5;
int rs = 6;
int re = 7;
```

In the above code snippet, rk(right knee) is connected at pin 3 on the driver, while rh (right hip) is connected to pin 2. With all the setup complete we moved on to actual movement.

3.1. Walking Algorithm

Walking: Our robot was balanced enough to stand on two legs. Because of this, we were able to lift two legs at a time without risk of falling over. Our walking motion was as follows: Two diagonal legs are lifted off the ground, then the corresponding hips or shoulders move forward. The same legs move down towards the ground and then move backwards. This creates a pushing motion towards the front. Lastly, the legs return to their original height at start position. The same motion is then repeated by the opposite legs. We

```
//START WALKING MOTION
     ////2 legs up
      rkpos(up);
      lepos(up);
      delay(time);
    //corresponding shoulder/hip
    //forward
      rhpos(up);
      lspos(up);
      delay(time);
    //2 legs down
      rkpos(down);
      lepos(down);
      delay(time);
    //corresponding
    //shoulder/hip back
      rhpos(down);
      lspos(down);
      delay(time);
    //legs return to
    //start pos
      rkpos(mid);
      lepos(mid);
      delay(time);
//repeat motion for opposite legs
```

included a delay variable in order to easily manipulate the robot's walking speed. Although not seen in this code snippet, the walking function also included a variable which counts the number of steps taken. One step was equal to the completion of the previously mentioned pushing motion by both sets of legs. This variable was later used in the navigation algorithm to traverse the course.

3.2. Turning Algorithm

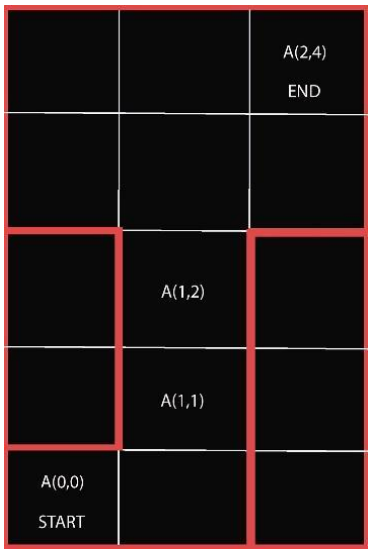
```
switch(Direction)
{
  case NORTH: Direction = WEST;break;
  case WEST:  Direction = SOUTH;break;
  case EAST:  Direction = NORTH;break;
  case SOUTH: Direction = EAST;break;
}
```

Turning: The physical motion for turning is the same as that of walking. The only difference is that only one set of legs does the walking motion. By having only one set of legs move, the robot pushed in only in one direction repeatedly. This resulted in the robot turning one way or the other. The most important thing to note is the code above. This code is executed whenever the turn function is called. Our robot must know what way it is facing at all times in order to not get lost in the course. When the robot turns either left or right the direction flag is changed to the new direction being faced. This will be covered in depth in the navigation algorithm.

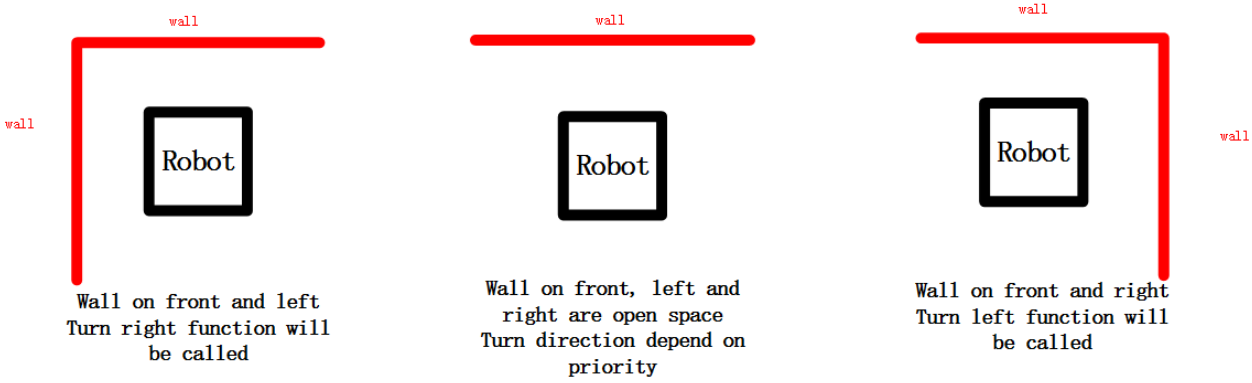
3.3 Navigation

As in the walking robot competition, we have known the starting point and ending point from the beginning. The obstacle locations were unknown to us however. As seen in the picture below, the starting point is grid (0, 0) and the ending point is grid (2, 4). We set our walking function in open loop, which means that it wouldn't stop walking until it detected an obstacle in front. In order to make sure our robot could get to the destination successfully, we used an error correct function to make sure robot was always in the center of grid during the walking. If the robot was too close

to the left or right side, the error correct function would be activated and shift the robot to the center of grid until the left/right side distance is larger than the preset value. Also, we set the walking step number in the main function in order to let the robot stop at the ending point.



When the robot detected a wall in front, it would call the turning function. If there was a wall on the left side and no wall on the right side, the robot would turn right. If there was a wall on the right side and no wall on the left side, the robot would turn left. In open space with no wall on the left or right side, we had different turning priorities for the robot because the ending point was set at the North-East of the stating point.



In order to make let our turning priority work effectively, we sign the direction to robot when turning function is called. For example, if the robot was facing the north

```
if(Direction == NORTH)
{
   startPosition();
   delay(20);
   LeftSensorval = analogRead(LeftSensorPin);
   RightSensorval = analogRead(RightSensorPin);
   BottomFrontval = analogRead(BottomFrontPin);
   RightAngledval = analogRead(RightAngledPin);
   LeftAngledval = analogRead(LeftAngledPin);
   Serial.println(LeftSensorval);
   delay(10);

  if(BottomFrontval < 250)
  {
      walk();
  }
  else
  {
   if(LeftSensorval < 180)
      left_Turn(8);
    else
      right_Turn(8);
   }
 }
```

direction at the beginning, the first turning priority is right if it detects an open space. When it turns right at the first corner, the facing direction changes to East and the first turning priority becomes left. Here is the code showing the turning when the robot faces North.

Conclusion

Rabbit Trails was successful in designing and operating the Walking robot. All the team members of Rabbit Trails worked so hard with building, testing and programming the walking robot. Our hardware team worked really close with the software team to support and make sure all the parts were working properly. In conclusion, we will discuss briefly the operation and specification of our walking robot.

We did a lot of research on mechanical design and made the decision to build a robot with 4 legs. We made one prototype sample leg and recorded all the dimensions for the mechanical design. We spent about a month on the design, manufacture and assembly of our robot. We put all the parts together and installed the battery at the end of March. After we finished testing the walking and climbing stair functions, we made decision to add bigger feet to stabilize its movement. For obstacle avoidance, the hardware team built the brackets to hold four sensors in the front and two side sensors. By using distance comparison between two font sensors

our robot was able to detect the difference between stairs and walls then call the walking or climbing function. For the side sensors, our robot was programed to read and compare the distance, then call the turn left or right function. In the last month, our team almost finished the climbing stair function by moving the left leg up to the stair then right leg, and using the two back legs push the whole body forward.

Specifications

- Dimension at rest position : 6.25 inches (W) x 6.75 inches (H)
- Power: based on the calculation our robot needs 1-1.5A for all the servo motors, microcontroller and sensors. We found rechargeable battery was really light weight: 3.2 Oz and large capacity: 7.4 V 2200mAh which could run about 90 minutes in walking mode. Please see the specification below for more information. For the voltage regulator, we used Polulu step-down voltage regulator requires input voltage: 4.5 V to 24 V and typical continuous output current: up to 3.5 A.

Capacity	2200mAh
Voltage	7.4V (peak at 8.4V)
Dimensions	2.69 x 1.45 x 0.7 inch
Weight	3.5 oz
Max. charge current 1C	2.2A
Max. discharge current 2.0C	4.4A

Cut off voltage	6V

- Arduino microcontroller: first we tried to use the UNO Arduino for controlling servo motors but it did not have enough Flash memory. We changed to Mega 2560 which has 256KB Flash memory, and higher IO/PWM 54/16. Please see the comparision table between UNO and Mega Arduino.

Name	Processor	Operating Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	Flash [KB]	UART
Uno	ATmega328	5 V/7-12 V	16MHz	6/0	14/6	32	1
Mega 2560	ATmega2560	5 V/7-12 V	16MHz	16/0	54/15	256	4

- Our servos operating voltage is 4.8V to 6V. No load speed: 0.12 seconds / 60° (4.8V) and Stall torque: 1.6 kg / cm (4.8V). Each servo draws less than 500mA. We used Adafruit 16-channel 12 bit PWM/Servo Shield servo controller. There's an I2C-controlled PWM driver with a built in clock. It is 5V compliant, which means you can control it from a 3.3V Arduino and still safely drive up to 6V outputs. The driver provides 6 address select pins so you can stack up to 62 of these on a single i2c bus. Adjustable frequency PWM up to about 1.6 KHz. 12-bit resolutions for each for each servo.
- Sensors and brackets: After couple weeks doing research and test difference kind of sensor, we made decision to use IR sharp sensor measuring

range: 10 cm to 80 cm. Operating voltage: 4.5 V to 5.5 V average current consumption: 30 mA, all the sensors get power directly from the Mega. We also used Solid Work to build and 3D print bracket which can hold 4 sensors in the font and 2 brackets for side sensors. The sensors are mounted about 1.5 inch above the ground for detecting wall.

We built our robot from scratch was a challenge for all of us because it was more about mechanical engineer. Our project building walking robot was completed on time. The walking function and navigating were checked and tested successful. When we added the navigating function, our robot was able to walk straight and checked for obstacle. We were so proud of our walking robot, if we have more time our robot would finish the stair climbing and descending easily.

Recomendations

After completing this project, we learned many things. First, we cannot understate the importance of the mechanical design. Purchasing a prebuilt robot was an option that we should have considered more seriously. The mechanical design of our robot was fundamental to the success of our algorithms. As a team of Electrical and Computer Engineers, we were not adequately prepared for this challenge.

With that said, more emphasis should have been placed on lowering the center of gravity of our robot. Because of its high center of gravity, balance was an issue that contstantly needed to be resolved. Also, we should have included an independent mechanism that allowed the robot to shift its weight without moving its legs. We

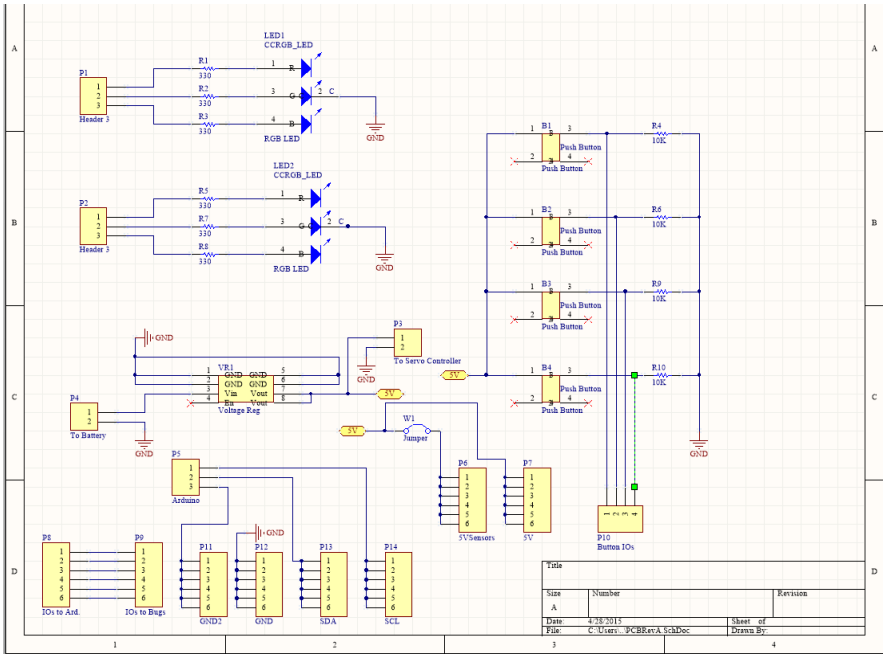
found that while climbing the stairs, movement would not occur unless the legs that were driving the motion had a significant amount of weight on them. Due to the awkward positions that our robot was forced into while climbing the stairs, moving them in order to shift the weight to a more appropriate leg was not always an option.

We should have designed our robot to be shorter and allowed him to take larger steps. It would have been helpful if the course parameters were more thoughtfully designed. The course was originally intended for micromouse and all the walls scaled accordingly. Attempting to naviage an obstacle course that contains obstacles significantly smaller than our robot proved awkward. Components available for purchase are only so small so this project would have been much easier and more meaningful if the robot had been built on a more human scale. Because of this, we should have done everything in our power to build the robot as small as possible.

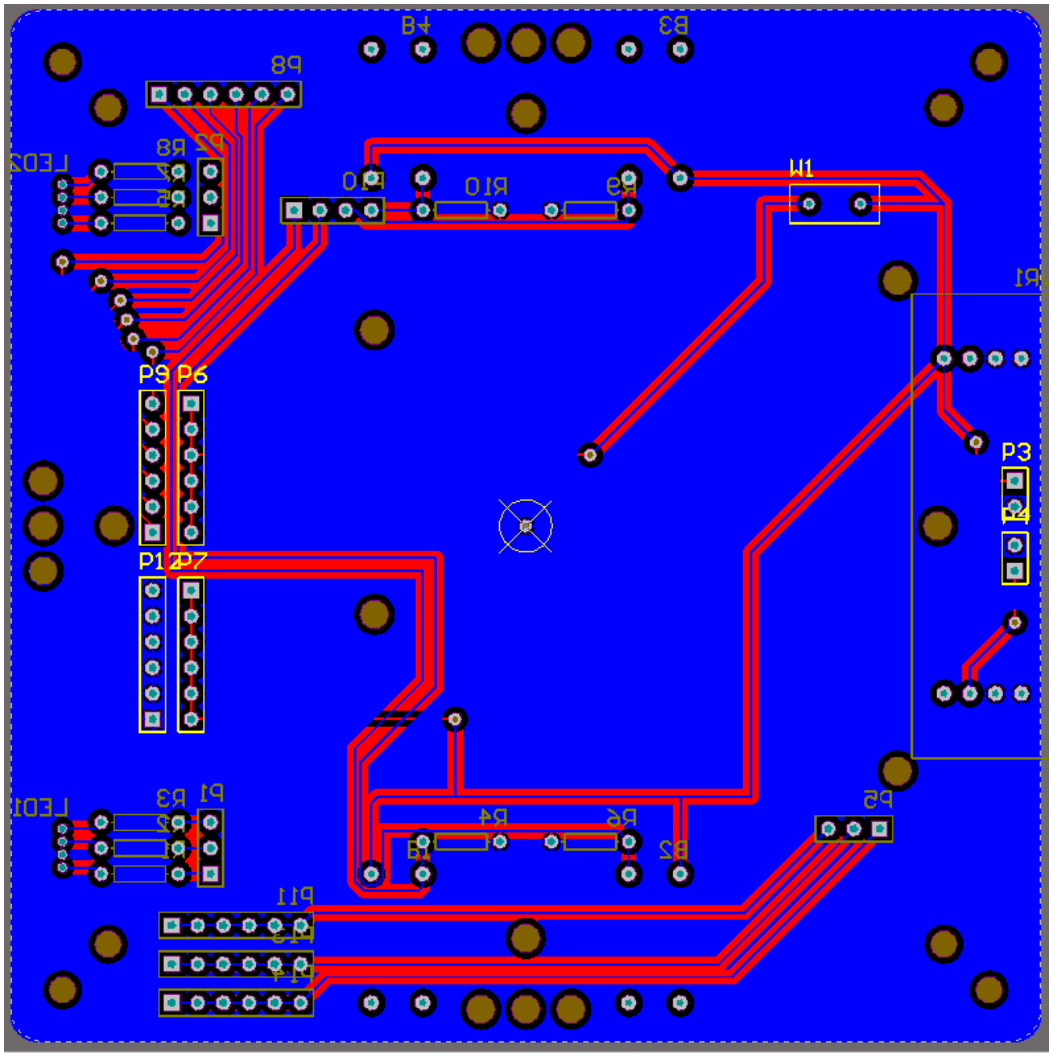
The timeline for the project was very short. Originally we planned to do all our design itteritavely. This approach usually yields the best result but setbacks forced us to use prototype designs in our final project. We should have forseen this and designed our prototypes with more care. Doing something right the first time is more valuable that proving a concept with a rough design. Overall, we learned a lot in participating in this competition, and we would all enjoy the opportunity to build a second robot, given the knowledge we now possess from the first. Hopefully, this project lives on at SDSU and furure participants can learn from our successes and from our failures.

Appendices

I.PCB Schematic



Push Button		B1, B2, B3, B4	PCBComponent_1	Push Button	4
RGB LED		LED1, LED2	RGB	RGB LED	2
Header 3	Header, 3-Pin	P1, P2	HDR1X3	Header 3	2
To Servo Controller	Header, 2-Pin	P3	HDR1X2	Header 2	1
To Battery	Header, 2-Pin	P4	HDR1X2	Header 2	1
Arduino	Header, 3-Pin	P5	HDR1X3	Header 3	1
5VSensors	Header, 6-Pin	P6	HDR1X6	Header 6	1
5V	Header, 6-Pin	P7	HDR1X6	Header 6	1
IOs to Ard.	Header, 6-Pin	P8	HDR1X6	Header 6	1
IOs to Bugs	Header, 6-Pin	P9	HDR1X6	Header 6	1
Button IOs	Header, 4-Pin	P10	HDR1X4	Header 4	1
GND2	Header, 6-Pin	P11	HDR1X6	Header 6	1
GND	Header, 6-Pin	P12	HDR1X6	Header 6	1
SDA	Header, 6-Pin	P13	HDR1X6	Header 6	1
SCL	Header, 6-Pin	P14	HDR1X6	Header 6	1
330	Resistor	R1, R2, R3, R5, R7, R	AXIAL-0.3	Res1	6
10K	Resistor	R4, R6, R9, R10	AXIAL-0.3	Res1	4
Voltage Reg		VR1	PCBComponent_1	Voltage Reg	1
Jumper	Jumper Wire	W1	RAD-0.2	Jumper	1



Ⅱ Sensor Experiment Data

***** Value in Table 1 & 2 is analogRead function return value***

Table 1: Compare Sensor Value with and without Capacity

Delay	10ms		1ms		100ms		10ms	
sample	10uF cap	no cap	10uF cap	no cap	10uf cap	no cap	22uf cap	no cap
1	310	309	310	309	332	309	308	314
2	308	308	309	309	309	322	309	311

3	309	309	308	309	308	308	309	311
4	309	319	315	309	309	307	309	310
5	308	308	309	310	308	308	309	308
6	308	309	308	310	308	309	301	310
7	308	300	320	317	315	308	320	309
8	309	325	311	308	321	309	309	309
9	308	335	309	308	309	300	331	308
10	308	309	308	300	308	309	316	308
11	310	313	319	309	309	309	310	309
12	309	311	309	309	309	307	310	309
13	320	308	308	309	309	310	311	309
14	331	309	301	327	314	309	311	309
15	315	308	310	308	316	327	309	308
16	309	310	309	309	308	309	309	310
17	313	309	309	308	308	309	309	321
18	310	309	310	315	309	307	309	327
19	308	308	309	334	308	309	307	334
20	309	308	323	311	309	311	309	310
21	309	308	317	309	330	316	307	315
22	308	309	312	309	304	300	308	309
23	308	309	311	310	309	307	309	309

24	309	308	310	310	308	309	313	309
25	309	304	308	308	309	308	308	308
26	308	309	308	309	310	309	304	308
27	309	325	309	309	315	310	320	308
28	309	334	308	309	321	309	308	309
29	308	310	308	307	309	307	331	308
30	308	309	308	308	308	308	312	308
31	322	309	308	302	312	308	310	308
32	309	310	308	323	322	310	309	309
33	332	310	320	335	311	310	310	308
34	317	309	309	310	314	312	310	308
35	311	308	309	309	320	300	308	304
36	309	310	311	308	309	308	309	310
37	309	307	311	309	309	309	309	323
38	310	310	310	309	309	307	308	332
39	310	308	308	308	310	308	309	313
40	309	310	308	308	309	313	309	310
41	309	308	309	309	335	332	308	310
42	308	310	308	309	300	308	309	309
43	309	309	309	309	309	307	308	311
44	309	300	308	308	308	309	309	310

45	308	322	309	308	309	309	308	310
46	309	310	308	309	311	309	302	308
47	309	335	317	334	309	315	320	308
48	309	313	331	310	323	309	309	310
49	308	313	313	309	309	308	330	317
50	300	310	309	310	308	307	314	307
avg	310.4	311.4	310.72	311.04	311.9	309.54	310.86	311.2
mode	309	309	308	309	309	309	309	308
st dev	5.394631	7.469667	4.874088	7.134166	6.860862	5.36508	6.171065	6.030535
min	300	300	301	300	300	300	301	304
max	332	335	331	335	335	332	331	334

Table 2: Sensor Value vs Diatance

Distance	*2.5inch*	*3 inch*	*3.5inch*	*4inch*	*5inch*	*6inch*	*7inch*	*8inch*	*9inch*	*10inch*
sample	*value*									
1	*652*	*607*	*533*	*478*	*394*	*338*	*294*	*266*	*242*	*247*
2	*650*	*607*	*536*	*478*	*394*	*335*	*294*	*266*	*243*	*221*
3	*651*	*604*	*533*	*504*	*398*	*354*	*294*	*266*	*244*	*222*
4	*659*	*604*	*533*	*478*	*395*	*338*	*294*	*266*	*243*	*236*
5	*651*	*609*	*537*	*477*	*395*	*336*	*294*	*287*	*242*	*223*
6	*651*	*605*	*533*	*481*	*401*	*329*	*294*	*266*	*241*	*221*

7	*663*	*605*	*535*	*480*	*395*	*340*	*295*	*265*	*242*	*208*
8	*649*	*612*	*539*	*479*	*394*	*336*	*294*	*279*	*241*	*221*
9	*648*	*604*	*535*	*478*	*401*	*336*	*295*	*266*	*241*	*220*
10	*656*	*605*	*535*	*483*	*396*	*340*	*294*	*265*	*242*	*210*
11	*648*	*638*	*570*	*479*	*396*	*336*	*294*	*256*	*241*	*223*
12	*650*	*605*	*537*	*480*	*404*	*337*	*295*	*269*	*244*	*224*
13	*650*	*604*	*534*	*484*	*396*	*336*	*295*	*266*	*243*	*221*
14	*650*	*640*	*564*	*480*	*394*	*337*	*295*	*264*	*243*	*226*
15	*649*	*605*	*534*	*481*	*424*	*337*	*294*	*267*	*243*	*222*
16	*682*	*605*	*534*	*486*	*394*	*337*	*295*	*265*	*242*	*221*
17	*650*	*639*	*561*	*480*	*394*	*337*	*295*	*265*	*242*	*225*
18	*649*	*609*	*534*	*480*	*416*	*337*	*294*	*271*	*242*	*221*
19	*684*	*607*	*534*	*490*	*394*	*337*	*295*	*266*	*242*	*222*
20	*649*	*611*	*535*	*479*	*394*	*337*	*295*	*266*	*269*	*236*
21	*649*	*611*	*536*	*478*	*389*	*337*	*295*	*274*	*245*	*222*
22	*655*	*605*	*533*	*509*	*395*	*338*	*292*	*265*	*243*	*222*
23	*649*	*607*	*534*	*478*	*394*	*337*	*298*	*265*	*256*	*247*
24	*651*	*607*	*538*	*478*	*390*	*337*	*294*	*295*	*245*	*222*
25	*655*	*605*	*536*	*508*	*398*	*339*	*294*	*264*	*243*	*223*
26	*652*	*604*	*536*	*479*	*395*	*337*	*296*	*264*	*228*	*237*
27	*652*	*610*	*542*	*479*	*396*	*338*	*294*	*287*	*245*	*224*

28	*650*	*605*	*534*	*502*	*401*	*337*	*294*	*267*	*243*	*222*
29	*646*	*605*	*534*	*480*	*394*	*337*	*301*	*265*	*232*	*230*
30	*650*	*610*	*538*	*480*	*395*	*337*	*295*	*281*	*243*	*224*
31	*649*	*605*	*533*	*478*	*397*	*337*	*293*	*265*	*241*	*222*
32	*660*	*605*	*533*	*483*	*394*	*338*	*298*	*264*	*241*	*212*
33	*652*	*613*	*540*	*479*	*394*	*337*	*293*	*280*	*243*	*223*
34	*651*	*605*	*534*	*479*	*400*	*337*	*293*	*267*	*244*	*223*
35	*651*	*605*	*534*	*478*	*394*	*337*	*319*	*267*	*242*	*221*
36	*650*	*605*	*565*	*479*	*394*	*337*	*293*	*256*	*247*	*222*
37	*649*	*605*	*534*	*479*	*425*	*338*	*293*	*266*	*241*	*222*
38	*684*	*606*	*534*	*478*	*395*	*345*	*312*	*265*	*241*	*221*
39	*650*	*608*	*564*	*479*	*396*	*337*	*294*	*265*	*246*	*225*
40	*649*	*607*	*534*	*479*	*421*	*336*	*293*	*270*	*242*	*221*
41	*688*	*608*	*533*	*478*	*397*	*365*	*284*	*266*	*242*	*221*
42	*653*	*608*	*535*	*480*	*395*	*336*	*294*	*266*	*271*	*226*
43	*650*	*608*	*534*	*481*	*416*	*336*	*293*	*271*	*242*	*223*
44	*682*	*608*	*533*	*480*	*398*	*359*	*288*	*265*	*241*	*222*
45	*648*	*608*	*534*	*480*	*396*	*336*	*294*	*265*	*259*	*251*
46	*649*	*607*	*535*	*480*	*391*	*336*	*293*	*270*	*245*	*222*
47	*655*	*608*	*534*	*481*	*399*	*353*	*293*	*265*	*241*	*222*
48	*652*	*608*	*533*	*480*	*396*	*338*	*297*	*264*	*256*	*239*

49	*650*	*608*	*535*	*480*	*392*	*337*	*294*	*294*	*243*	*223*
50	*653*	*607*	*533*	*480*	*395*	*330*	*294*	*264*	*243*	*222*
avg	*654.5*	*608.72*	*537.82*	*481.94*	*398.02*	*338.58*	*294.94*	*268.58*	*244.12*	*224.52*
mode	*650*	*605*	*534*	*480*	*394*	*337*	*294*	*265*	*242*	*222*