Design Competition 2010 Project Report

Edward Jen

DSGN 360
Professor Michael Peshkin
06/09/2010

[The project report deals with the robot built for the Northwestern University Design Competition 2010 by Team Fork – Edward Jen, Sourya Roy, Nicolas Renold]

Table of Contents

1. Introduction 4

1.1 Arena 4

1.2 Tokens 5

2. Design Requirements 6

3. Physical Design Overview 7

3.1 Electrical Subsystem 8

3.1.1 PIC Microcontroller 9

3.1.2 PWM Motor Driver 10

3.1.3 Power Regulator 10

3.1.4 Control Switches 10

3.1.5 7-Output Optical Sensors 11

3.1.6 Bump Sensors 12

3.1.7 Motors 12

3.2 Mechanical Subsystem 13

3.2.1 Base Holder 14

3.2.2 Token Collector 15

3.2.3 Wheel Assembly 15

3.2.4 Magnetic Attachment 16

4. Software Design 17

5. Competition Outcome and Final Thoughts 18

6. Acknowledgements 19

Appendix A: Tournament Rules 20

Game 20

Arena 20

Tokens 21

Robot 21

Referees 21

Appendix B: Instructions on using an H-Bridge for PWM 22

Appendix C: Building an IR Sensor Circuitry 23

Appendix D: Photographic Representation of the Robot Assembly 24

Appendix E: PIC-C Source Code 27

Configuration Header File 27
Main Program Routine 28
PWM Motor Driver 33
Sensor Input Driver 35
References 37

1. Introduction

The Northwestern University Design Competition is a yearly robotics competition held in the spring where student built autonomous robots to compete against each other in a given challenge.

In this year's design competition, robots competed in a tournament where they had to collect metal and wooden tokens distributed along snaking curves painted on an eight-foot-square arena. The competition pits two teams of robots at a time against each other in an arena to see which one can collect the maximum number of tokens to win the match. The robots start on opposite corners of the arena. Each round is of two minutes. The team who has the maximum number of points after the round advances to the next.

1.1 Arena

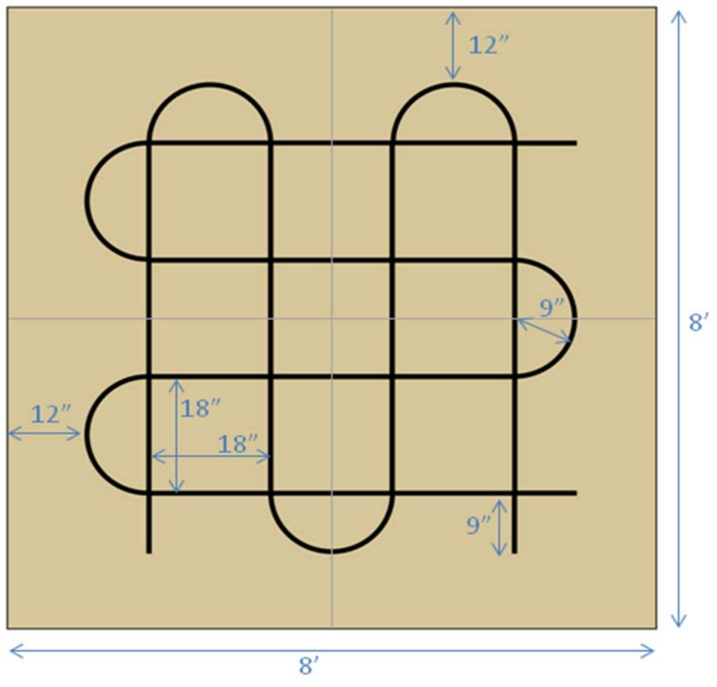


Fig. 1: *Competition Arena*

The arena shown in Figure 1 consists of an 8'x8' masonite base, which is in turn composed of 4 pieces, each 4'x4'. The arena is supported by a wooden frame such that it sits a few inches off the floor. The arena is painted with 3/4"line curved tracks on the base in contrasting colors as shown in the figure. Both of the tracks will extend 9" past the last intersection at the start and end. There is also a 1½" wall around the perimeter of the base to prevent aggressive robots from knocking their opponents out of the ring.

1.2 Tokens

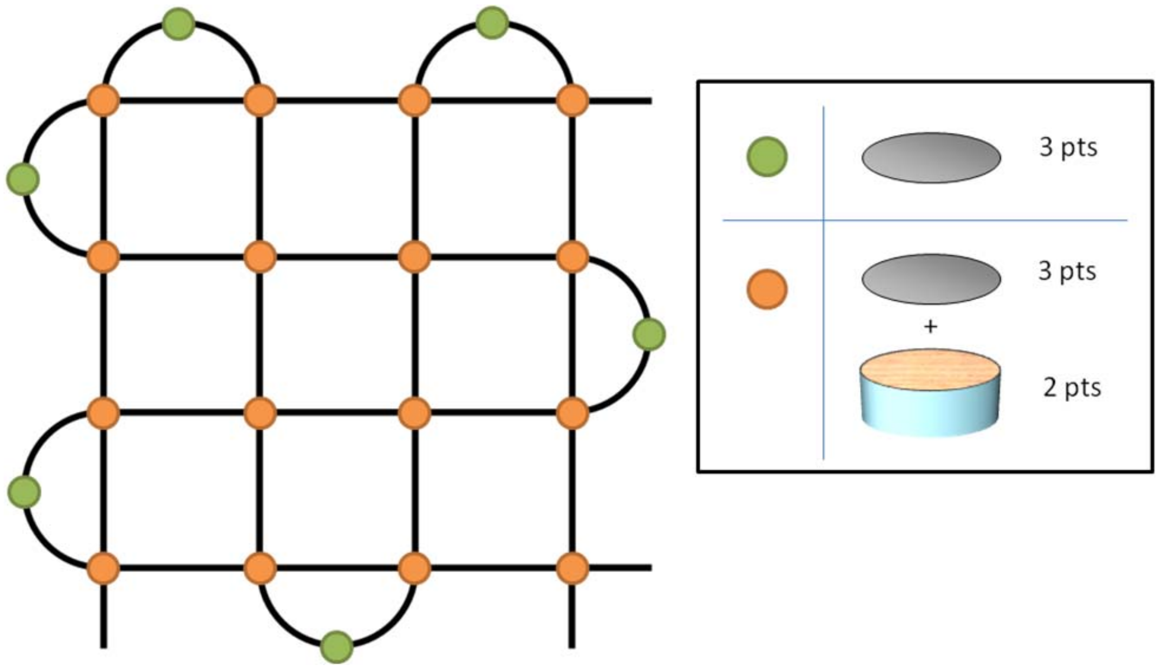


Fig. 2: *Arrangement of Tokens in the Arena*

According to the above figure, both metal and wood tokens are present on the board. Each wood token has a metal one resting on top of it. Metal tokens are also located at the top of each arc. There are 16 wooden tokens and 22 metal tokens in total.

Each round is for 2 minutes and the robot gets +2 points for collecting each wooden token and +3 points for each metallic token.

[Please refer to ***Appendix A*** *for the tournament rules.]*

2. Design Requirements

In order to participate in the competition, we had to build a robot according to the following criteria specified by the competition committee:

1. Robots must fit the size limit of 12” x 12” base (no height limit)

2. Only 2 drive motors allowed – must be provided Faulhabers

3. Weight limit: none
    - This should be sufficiently constrained by the drive motors.

4. Only one magnet allowed per robot (either electromagnet or permanent magnet is fine)
    - The magnetized collection surface may be no larger than 2”x1”

5. Robots must not actively interfere with the sensors of other robots

6. No use of fire or explosive materials. Compressed air may be used if pressures and materials are such that no hazard is created. This is intended to prevent CO2 cartridges and the like, but not to prevent using air to convey energy or move balls.

7. Robots must not lift other robots

8. Robots may not endanger the arena or other robots or people.

3. Physical Design Overview

In accordance with the above mentioned guidelines, we came up with our robot for this competition, the BigFork:

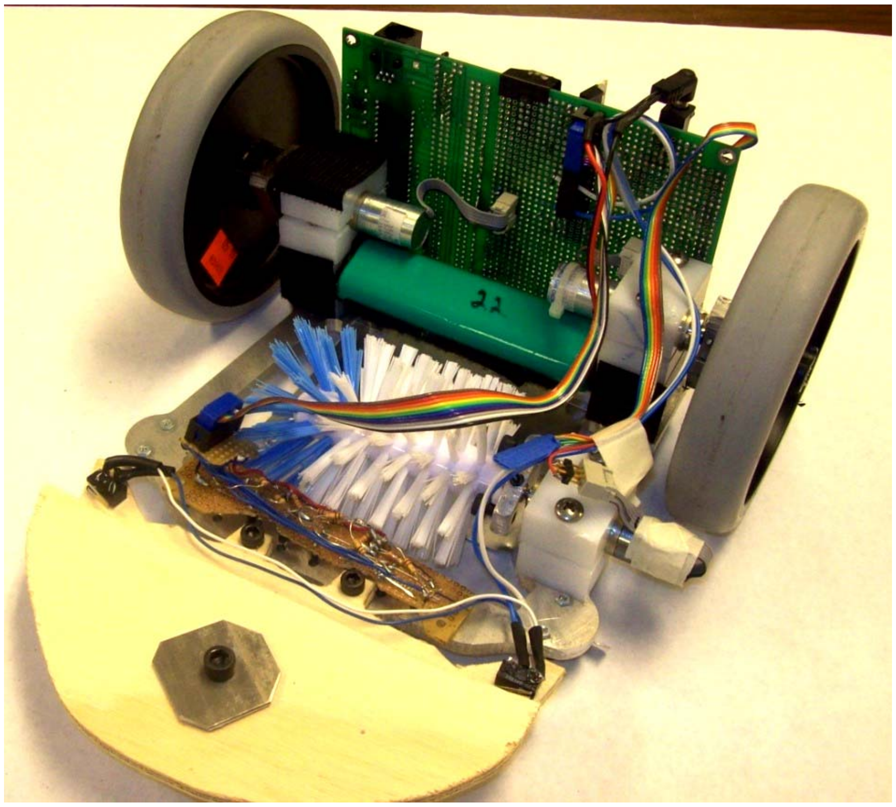


Fig. 3: *Angular view of BigFork*

We chose a compact design approach this time to build our robot, having all the electronic functionality integrated into a single circuit board. The compact design approach was simple to design and easy to debug.

The design can be divided into two major subsystems - electrical and mechanical.

3.1 Electrical Subsystem

The electrical subsystem consists of the following components:

1. PIC Microcontroller
2. PWM Motor Driver
3. Power Regulator
4. Control Switches
5. 7-Output Optical Sensors
6. Bump Sensors
7. Sweeper Motor
8. Left & Right Motors

The PIC microcontroller gets inputs from the 7-output optical sensors and the bump sensors and outputs the necessary PWM waveform to the PWM motor driver to run the motors. The following figure shows the interconnected module layout:

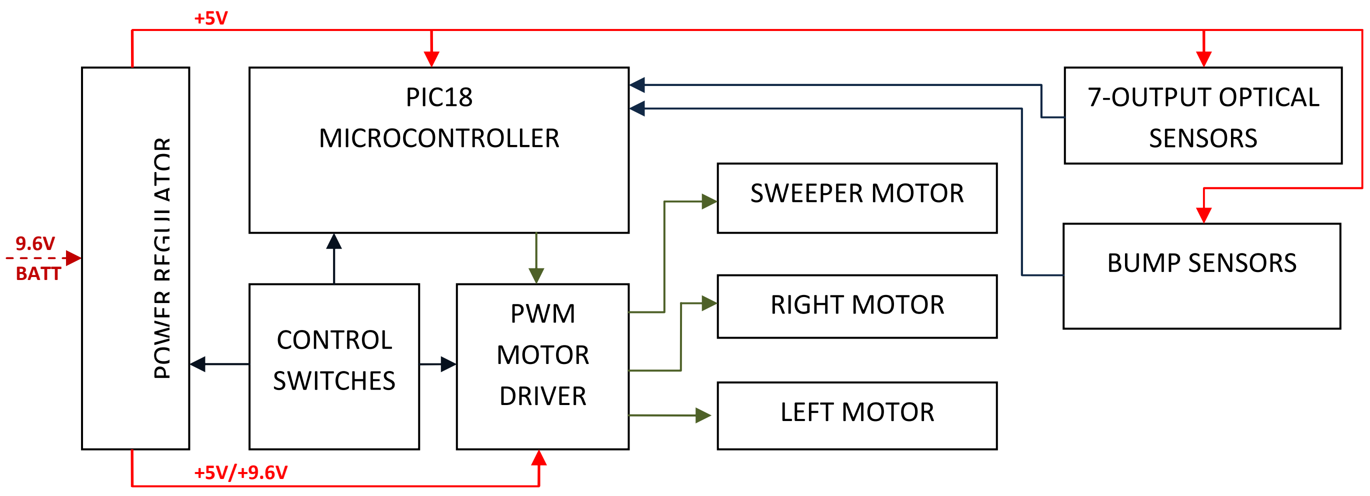


Fig. 4: *Interconnected Module Layout*

Other than the above mentioned modules, the electrical subsystem also consists of the following components:

4. A 9.6V NiMH 4500mAh rechargeable battery pack to power the robot.
5. Socket headers to interface the external components (the motors and the sensors)
6. Multi-colored ribbon cables fitted with 10-pin IDC socket for interconnection.

The PIC microcontroller, PWM motor driver, the power regulator and the control switches are housed in a single circuit board as shown in Figure 5. This is in turn connected to a 9.6V battery, three motors and to the optical and bump sensors.

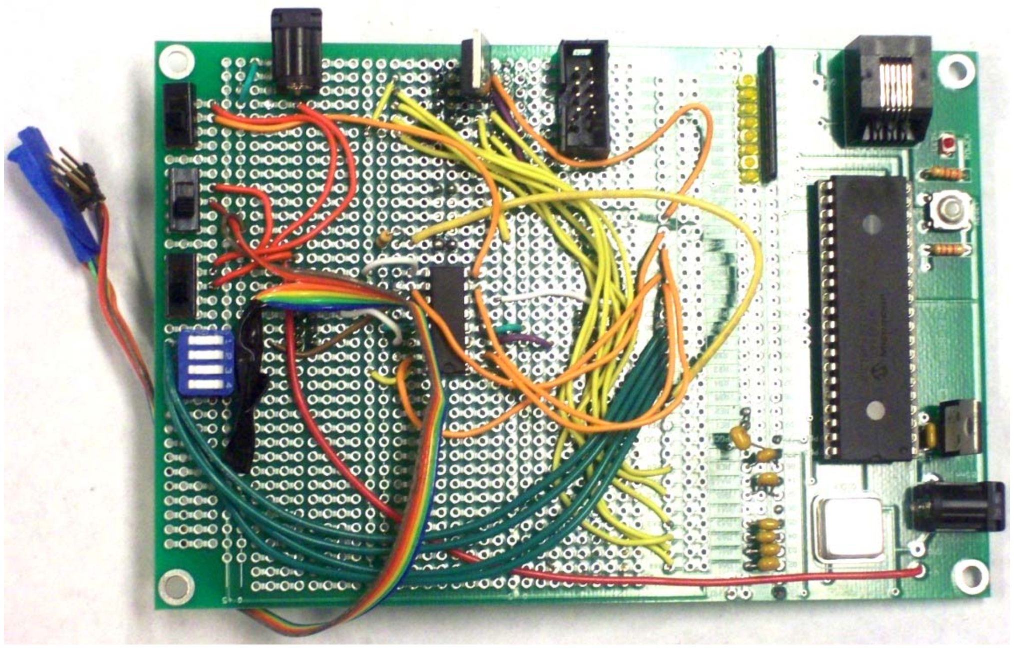


Fig. 5: *Circuit Board*

3.1.1 PIC Microcontroller

The main circuit board contains a PIC18F4520 microcontroller. The 7 output optical sensors are connected to the ADC inputs of the PIC while the bump sensor is connected to a digital input. The PIC runs the control program and generates the necessary PWM waveform to drive the motors. Figure 6 shows the pin-outs for the PIC microcontroller and Table 1 shows the pin connections.

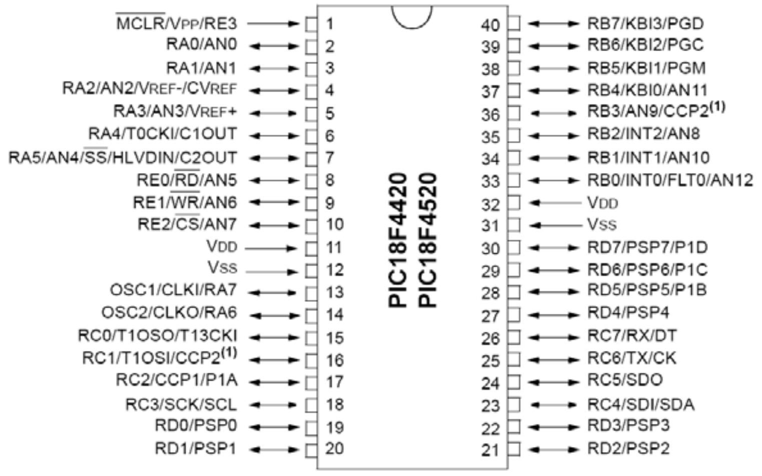


Fig. 6: *PIC18F4520 Pin Outs*

PIN	CONNECTION
RC0	ENCODER_LEFT_A
RC1	PWM1 (Right Motor)
RC2	PWM2 (Left Motor)
RC3	ENCODER_LEFT_B
RC4	ENCODER_RIGHT_A
RC5	PWM2 (Left Motor)
RC6	PWM1 (Right Motor)
RC7	ENCODER_RIGHT_B
RB0	SWEEPER MOTOR EN1
RB1	SWEEPER MOTOR EN2
RB3	BUMP SENSOR
RA0	OPTICAL0
RA1	OPTICAL1
RA2	OPTICAL2
RA3	OPTICAL3
RA5	OPTICAL4
RE0	OPTICAL5
RE1	OPTICAL6

PIN	CONNECTION
RD0	STATUS LED
RD1	STATUS LED
RD2	STATUS LED
RD3	STATUS LED
RD4	STATUS LED
RD5	STATUS LED
RD6	STATUS LED
RD7	STATUS LED

LEGEND

🟫	MOTOR ENCODER
🟧	PWM MOTOR DRIVER
🟥	SWEEPER MOTOR
🟩	OPTICAL SENSOR
🟨	BUMP SENSOR
⬜	STATUS/DIAG. LEDS

Table 1: *Pin connections to the PIC*

3.1.2 PWM Motor Driver

The PWM motor driver essentially consists of 2 dual H-Bridge ICs, which can drive 4 motors using PWM. However, in this implementation we use one H-Bridge IC to drive the left and right wheels using the PWM signals from the PIC. The second IC only runs the sweeper motor (bidirectional) using two input enable signals. The motor driver is connected to a 9.6V/5V combined power output from the power supply. It uses 9.6V bus to drive the motors while the 5V bus drives the logic.

*[Please refer to **Appendix B** for instructions on using an H-bridge for PWM.]*

3.1.3 Power Regulator

The robot is powered by a 9.6V NiMH battery. The motors run at 9.6V for optimal speed, but the logic circuit and sensor components require a stable 5V input. This is implemented using a LM7805C 5V voltage regulator IC. The power regulator circuit also includes a fuse, indicator LED and three switches (one to power up the system and two other for wheel and sweeper motors respectively). The switches allow programming/debugging without turning on the motors.

3.1.4 Control Switches

This circuitry contains three 2-way switches and a 4-in-1 DIP switch. The 2-way switches are used to power up the system and the wheel and sweeper motors, where as the 4-in-1 DIP switches are connected directly to the PIC. This was done to customize the behavior of the PIC software on the fly (by selecting a specific switch configuration) rather than re-flashing the system again.

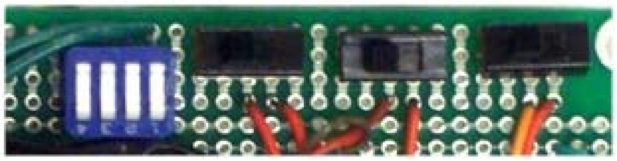


Fig. 7: *Control switches on the circuit board*

3.1.5 7-Output Optical Sensors

There are eight infra-red optical sensors in the front of the robot to detect the line. This is the core component for a line-sensing PID motion control. Each sensor essentially contains an IR LED and a phototransistor. Since the arena is covered with dark curved lines, the phototransistors detect the reflected IR beam when there is no line in sight to absorb the reflection. A non-detection of the beam by the phototransistor implies that the sensor is over a dark area (i.e. part of a line). Though this can be done by using only 3 sensors, we are using 8 to increase the resolution and smoothing out the PID response.

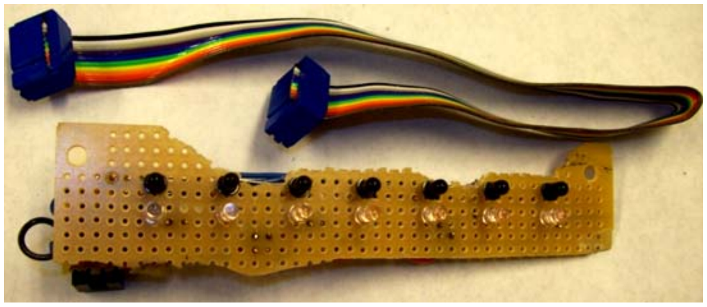


Fig. 8: *Optical sensor circuit with connector ribbon cables*

[Please refer to ***Appendix C*** *for instructions on building an IR sensor circuitry.]*

3.1.6 Bump Sensors

The bump sensors are attached to the front of the robot. They are basically push-switches and send logic high when they are pressed by a wall or an enemy robot.

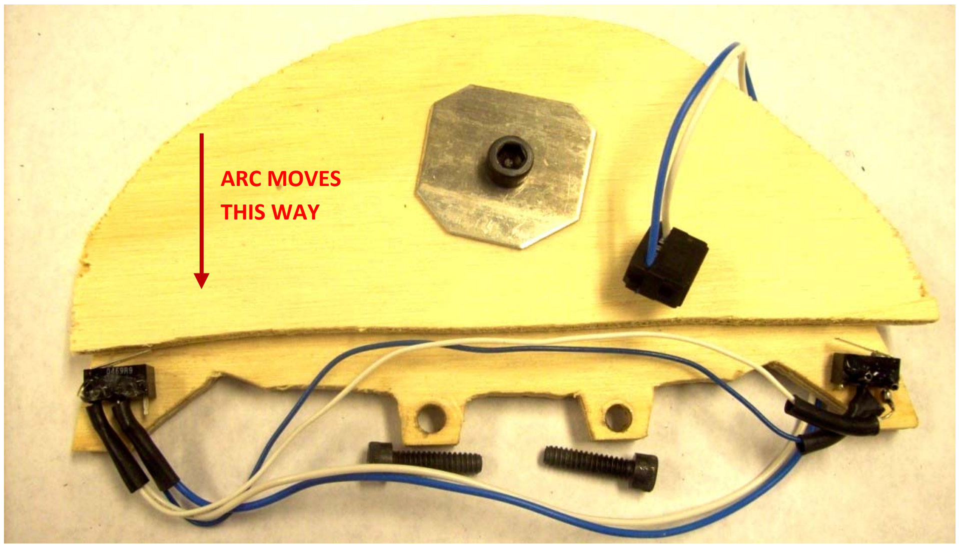


Fig. 8: *Bump Sensors*

3.1.7 Motors

We use three Faulhaber 1524E006S motors with 141:1 gearhead motors to run the two wheels and a token sweeper. These motors also contain a built in HES164A quadrature magnetic motion encoder. These motors are rated at 6-12V and have around 140 RPM at 9.6V. The quadrature encoder gives 141 x 4 = 564 counts/rev at output shaft in a 4x decoding mode. The wheels and the sweeper are explained in detail in Section 3.2 (Mechanical Subsystem).

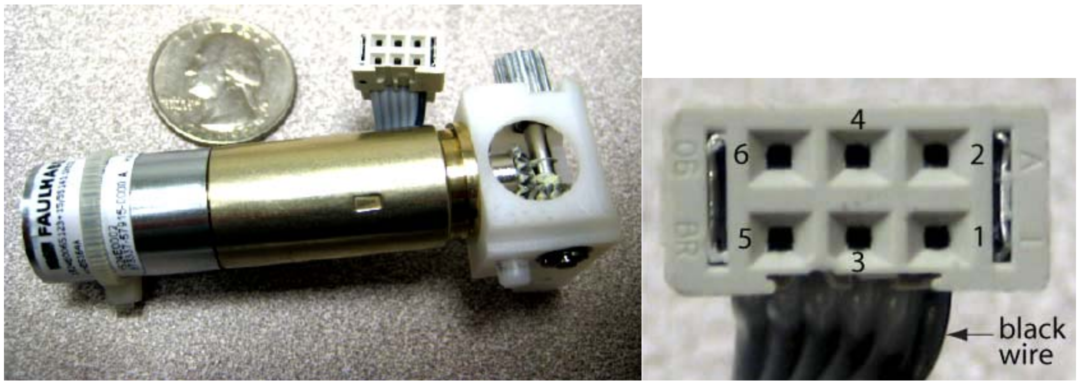


Fig. 9: *Faulhaber Motor & Connector*

However, the motor uses a non-standard connector. The pins on the connector are (see figure above): (1) Motor +, (2) +5V to power the encoder, (3) Encoder channel A, (4) Encoder channel B, (5) GND for the encoder, (6) Motor -.

3.2 Mechanical Subsystem

The mechanical subsystem is divided into the following components:

1. Base Holder
2. Token Collector/Sweeper Motor
3. Wheel Assembly

The following figure illustrates the mentioned components:

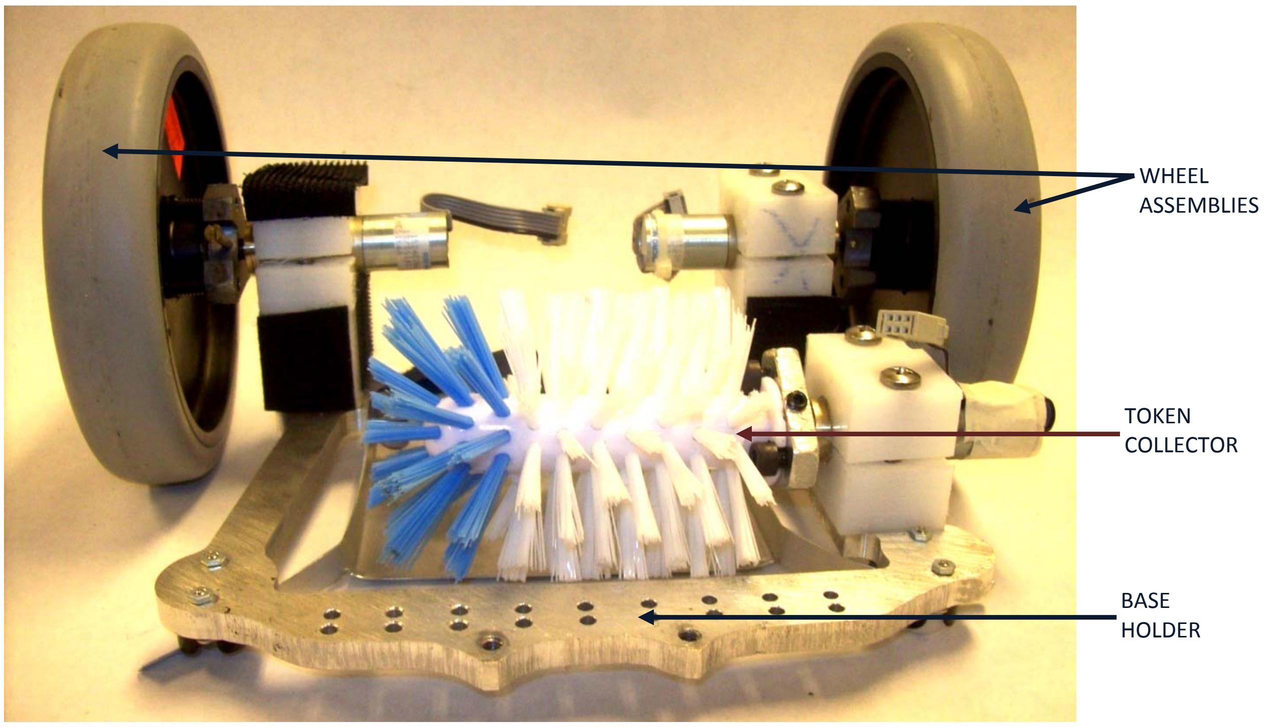


Fig. 10: *Mechanical Subsystem*

3.2.1 Base Holder

The base board essentially contains all the other components - the wheel motors, battery, token collector (sweeper motor), optical & bump sensors, and the PCB. It also has a magnet attached at front to collect the metal tokens, and an enclosure to store the collected tokens. The aluminum chassis was designed in Soldiworks with a CNC-aluminum laser frame. The front holes are for the photodiodes and IR LED emitters for the line sensor. The front two holes are for magnet attachment. The holes are for the motor plastic block attachments and sweeping motor attachments.

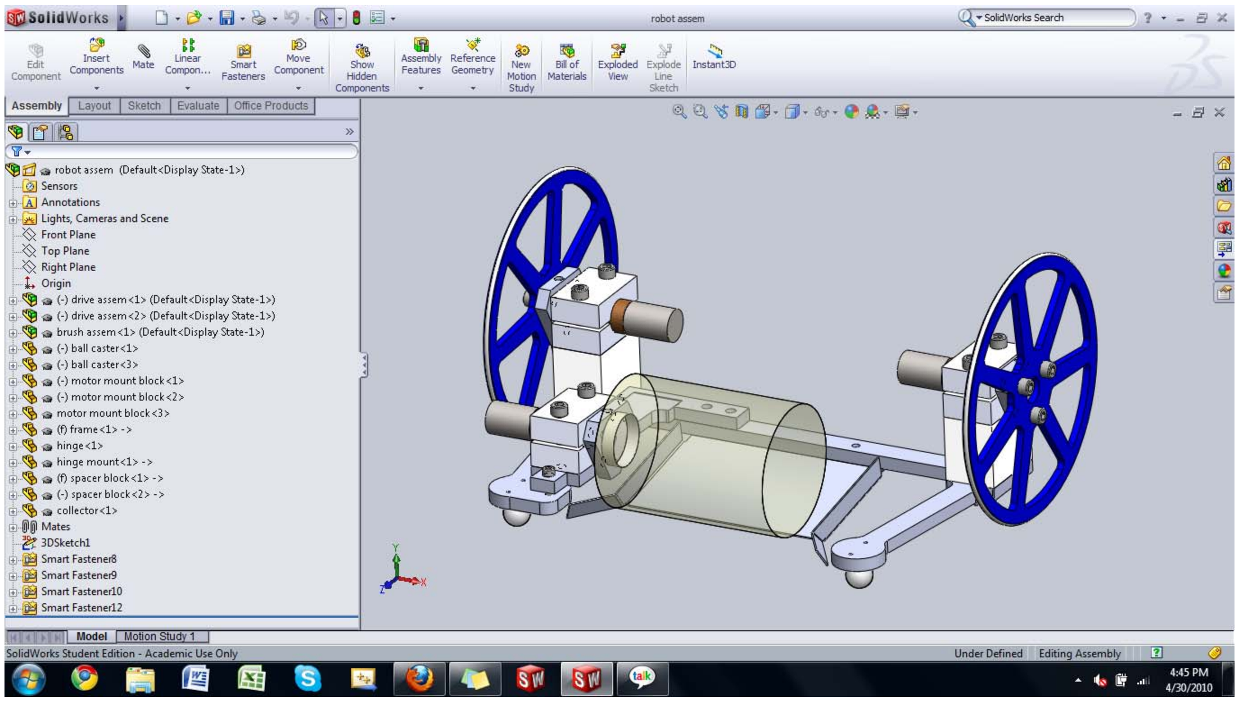


Fig. 12: *SolidWorks CAD model of the base holder with wheels and token collector on Windows 7 (courtesy of Nick Renold)*

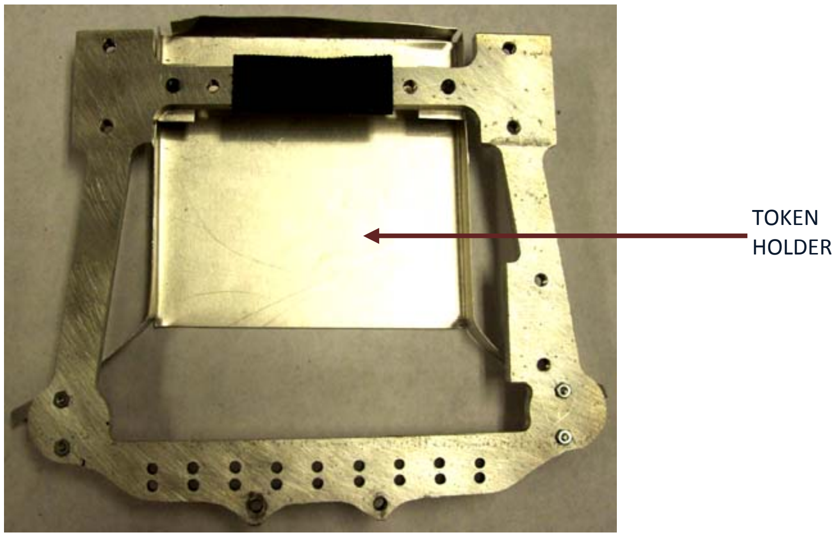


Fig. 13: *Cut-out base holder from the above CAD model*

3.2.2 Token Collector

The token collector consists of a motor with a rotating brush which sweeps the tokens which comes in the way of the robot into the token holder enclosure. The brush can be run bi-directionally. It runs inwards while collecting tokens during the game-play and outwards during the removal of the collected tokens from the system. The token storage device can be seen in Figure 12.

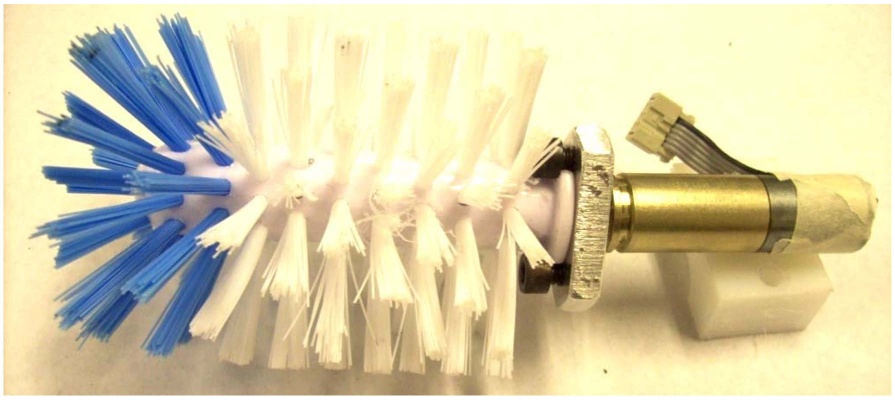


Fig. 14: *Token Collector*

3.2.3 Wheel Assembly

We used heavy duty trolley wheels for the robot. However, in order to fit the narrow shaft of the motor to the big wheel, we designed an intermediate wheel-motor connector. The wheels with the motors are in turn connected to the base holder using plastic enclosures.

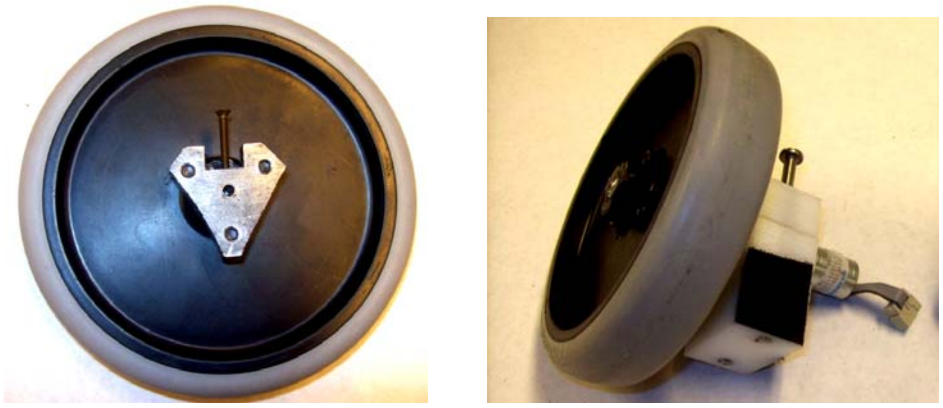


Fig. 15: (Left) *A wheel with the intermediate connector*
(Right) *The wheel assembly including the wheel, motor, and plastic enclosure*

3.2.4 Magnetic Attachment

The magnet attachment is also machined out of aluminum here, and the two holes on the top attach to the chassis of the robot. As magnetic tokens pass under the magnet, the magnet will automatically pick them up. Any left over metal tokens and wooden tokens will be swept up by the toilet sweeper.

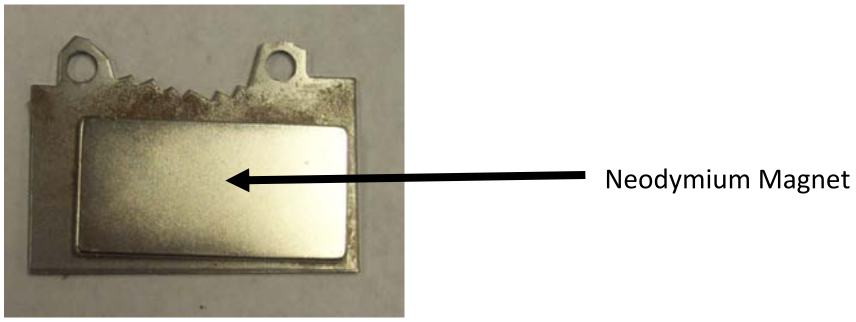


Fig. 16: *Neodymium Magnet with Aluminum*

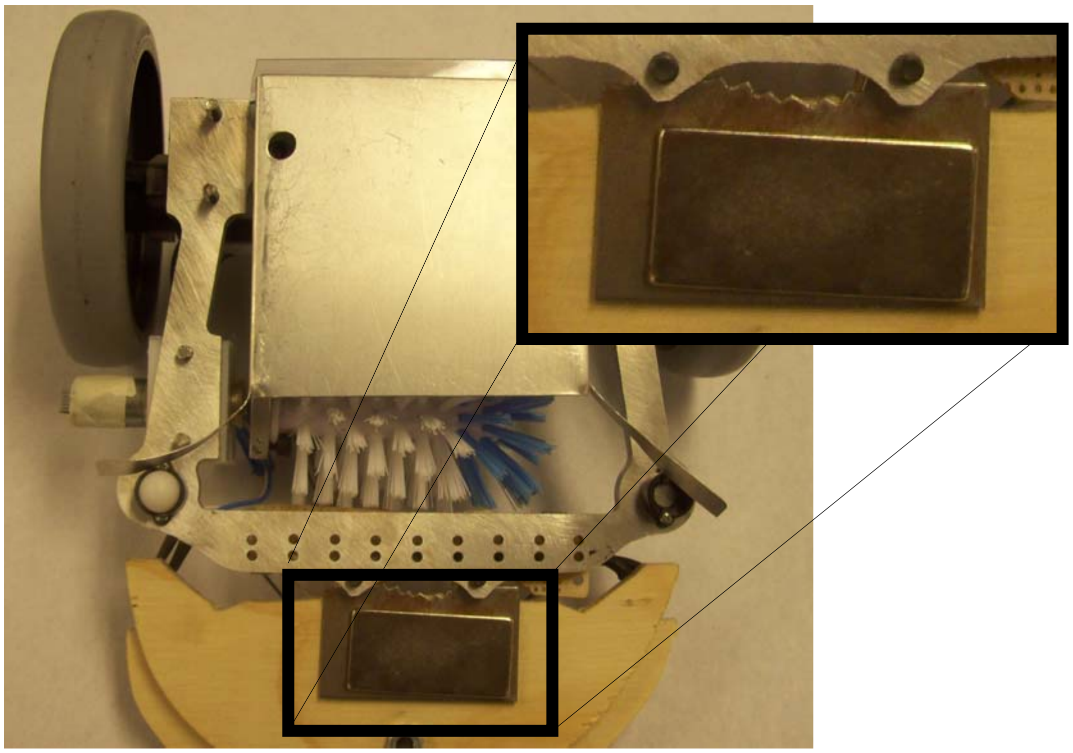


Fig. 17: Magnetic *Attachment to Aluminum Chassis*

[Please refer to ***Appendix D*** *for a photographic representation of the robot assembly process.]*

4. Software Design

The software code is written in C and compiled using the CCS PIC-C compiler. The algorithm for driving the robot generally consists of the following steps:

```
while(TRUE) {
     if (center sensor input is high)   //robot following the line
          go straight;
     else                                //center is not on line
          calculate position value (mPos) using sensor readings;

          //PID Calculations
          Diff = tPos - mPos;  //P
          Prop = Diff * Kp;

          Integral += Diff; // I
          Integral *= Ki;

          Rate = Diff - Difflast; // D
          Deriv = Rate * Kd;

          Difflast = Diff;

          Control = Prop + Deriv + Integral;

          if (control < 0) //robot banking right
               turn left depending on magnitude of control;
          else            //robot banking left
               turn right depending on magnitude of control;


          if (bumper sensor == 1){ //obstacle detected
               go forward for 1.5 seconds; //clear movable obstacle
               move backward for 2 seconds;//go back, avoid collision
               rotate;
               reset mPos value and go forward;
               start forward thrower;
          }
     }

}                             //repeat the whole loop
```

*[Please refer to **Appendix E** for the full software code for the PIC controller driving the robot.]*

5. Competition Outcome and Final Thoughts

The competition was a great way to showcase our robot and design methodology. The mechanics of the robot was extremely strong with the light and strong aluminum frame. On the final competition day, our robot played five rounds and successfully moved to the quarter finals but unfortunately could not qualify for the semi-finals. We think that the performance of our robot was degraded due to a software glitch in the PID driver which gave the robot some abrupt oscillations while crossing through some portions on the curve. We plan to fix these problems in our next robot for the coming year's design competition.

6. Acknowledgements

I, on behalf of my team would like to thank the following persons for their help and guidance relating directly or indirectly to the competition:

1. Professor Michael Peshkin, *Dept. of Mechanical Engineering*
2. Professor Alan Sahakian, *Dept. of Electrical Engineering and Computer Science*
3. Professor Allen Taflove, *Dept. of Electrical Engineering and Computer Science*
4. Professor Martin Plonus, *Dept. of Electrical Engineering and Computer Science*
5. Rick Marzec, *Senior Support Technician, Dept. of Mechanical Engineering*
6. Steve Jacobson, *Machine Shop Instructor, Segal Design Institute*

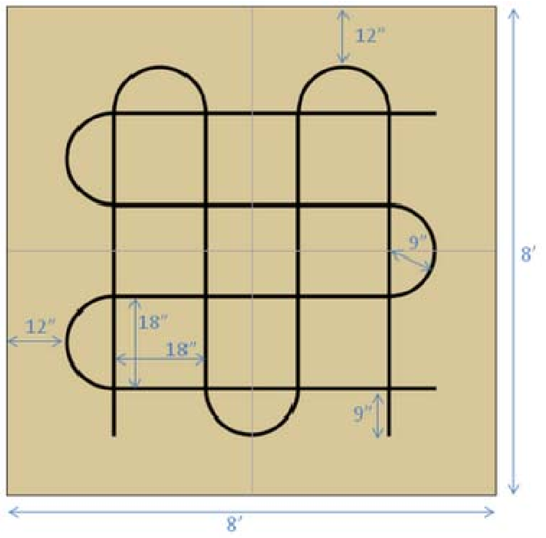
Appendix A: Tournament Rules

[Source: http://sites.google.com/site/dcnuinfo/2010-competition-rules]

Game

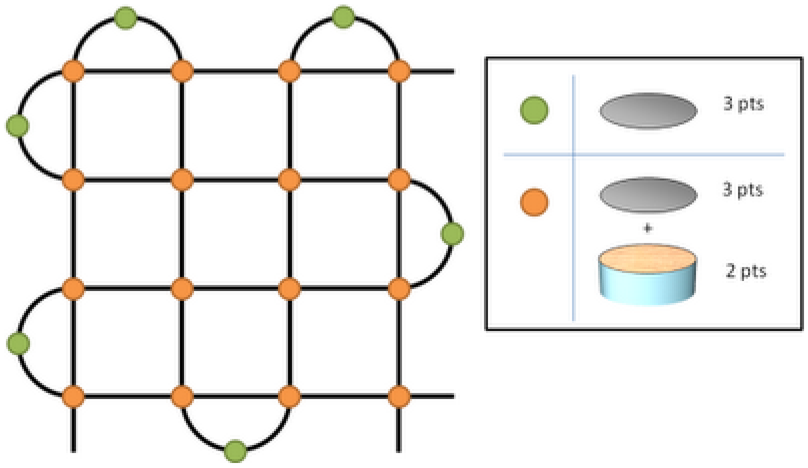
- 2 minute rounds
- Minimum of 2 rounds per robot.
- There will be a seeding round to determine competition bracket.
- The robot with the most points wins.
- Teams may clean or dust the arena with a dry rag (provided) at the beginning of the round.
- It is difficult to distinguish intentional aggression from bad programming, so both will be allowed.
- Teams may turn off their robots at anytime but they must remain off for the remainder of the round

Arena



- 8'x8' masonite base
  - o Composed of 4 pieces, each 4'x4'
    - § Will make transportation easier
    - § Since we have to have seams somewhere, this guarantees both teams have to deal with the same amount of seam
  - o Supported by a wooden frame so it sits a few inches off the floor
- 3/4"lines will be painted on the base in contrasting color
  - o The track will be centered, leaving 1' of clearance to the wall
  - o Each square of the grid will be 18" on its side
  - o The curves will be half-circles of radius 9"
  - o Both of the tracks will extend 9" past the last intersection at the start and end
- 1½" wall around the perimeter of the base
  - o Prevents aggressive robots from knocking their opponents out of the ring
  - o Will have a 1" strip of retro reflective tape on the top of the wall

Tokens



- Both metal and wood tokens will be present on the board.
- Wood
  - Worth 2 points each
  - 16 total – one located at each intersection
  - 1" diameter cylinders; 3/8" high
    - Most line following sensors should be able to pass over it
- Metal (steel)
  - Worth 3 points each
  - 22 total
    - One resting on each wood token (16)
    - One located at the top of each arc (6)
  - 1" diameter circles; nearly flat

Robot

- Size limit 12" x 12" base (no height limit)
- Only 2 drive motors allowed – must be provided Faulhabers
- Weight limit: none
  - This should be sufficiently constrained by the drive motors.
- Only one magnet allowed per robot (either electromagnet or permanent magnet is fine)
  - The magnetized collection surface may be no larger than 2"x1"
- Robots must not actively interfere with the sensors of other robots
- Robots must not damage or deface the arena
- Robots must not lift other robots

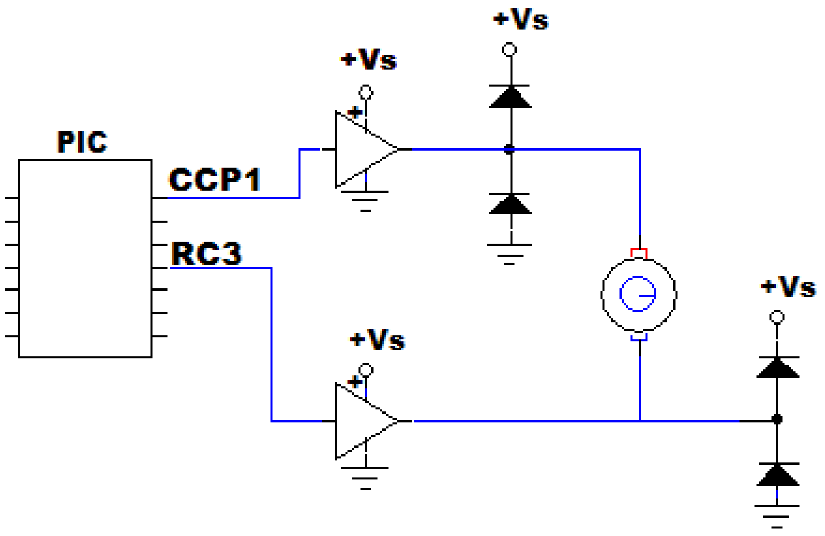
Referees

There will be two volunteer referees observing and timing each round. The volunteers may be participants in DC, but may not be involved with either of the competing robots.

Appendix B: Instructions on using an H-Bridge for PWM

[Adapted from http://hades.mech.northwestern.edu/wiki/index.php/Pulse_width_modulation]

In use with a PIC, two of the half-bridges are connected to two outputs of the PIC. One output is duty-cycle adjusted by the PIC's PWM capability, and the other output is held at a steady logic high or logic low. The following shows a circuit schematic for connecting two half-bridges to the PIC and a motor.



Here we've used CCP1 as the PWM output, and RC3 as the steady logic output. When both CCP1 and RC3 are high, +Vs is switched to both sides of the motor and so there is no voltage across the motor. Similarly when CCP1 and RC3 are both low. When RC3 is low and CCP1 is at 50% duty cycle, there is a voltage of +Vs across the motor half the time, and it spins. If CCP1's duty cycle is 75%, the motor spins faster, because +Vs is across the motor three-quarters of the time. If RC3 is high and CCP1 is at 75% duty cycle, there is a voltage of +Vs across the motor only one-quarter of the time, and it is reversed in polarity from the previous example. Thus the motor spins slowly, and in the opposite direction. You can put the above pieces together to torque the motor in either direction with any duty cycle between 0 and 100%. The figure on the side shows connection

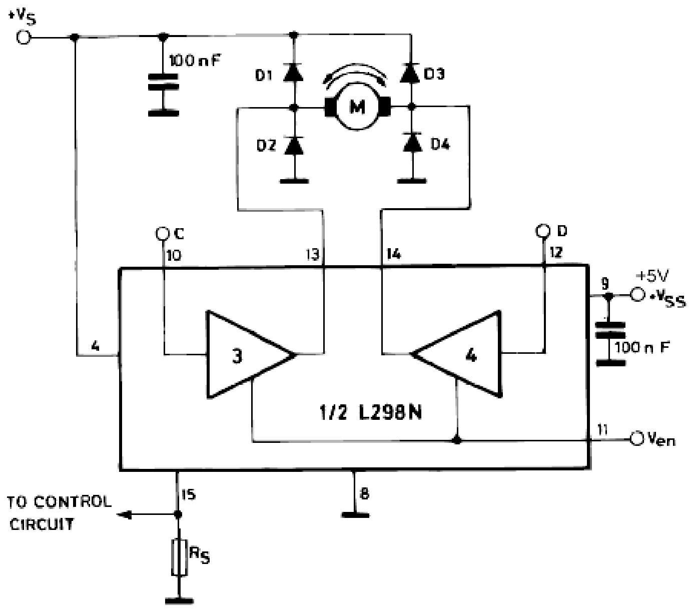
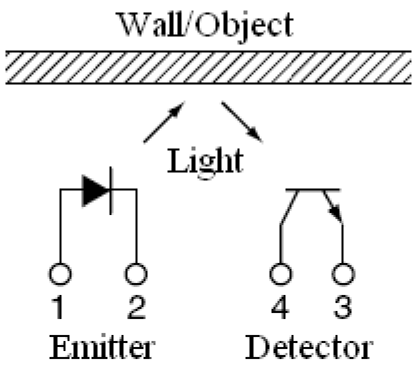


diagram from the datasheet of the L298 H-bridge (the type used in building our robot).

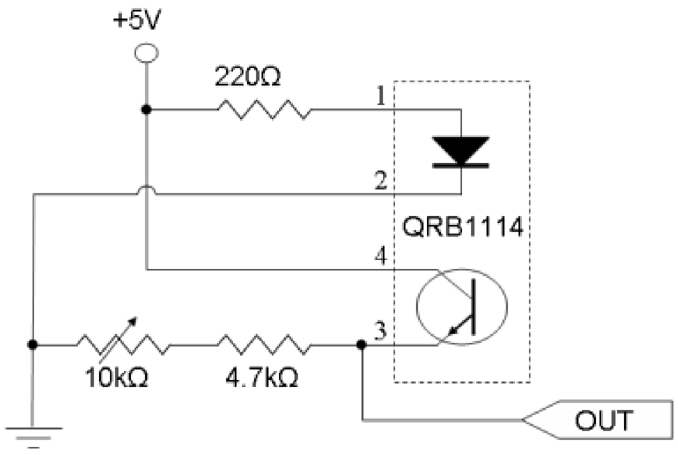
Appendix C: Building an IR Sensor Circuitry

[Adapted from: http://hades.mech.northwestern.edu/index.php/Optoreflector]

Optoreflector sensors contain a matched infrared transmitter (LED) and infrared receiver pair. These devices work by measuring the amount of light that is reflected into the receiver. Because the receiver also responds to ambient light, the device works best when well shielded from ambient light, and when the distance between the sensor and the reflective surface is small (the graph below shows how distance affects the output value). IR reflectance sensors are often used to detect white and black surfaces. White surfaces generally reflect well, while black surfaces reflect poorly.



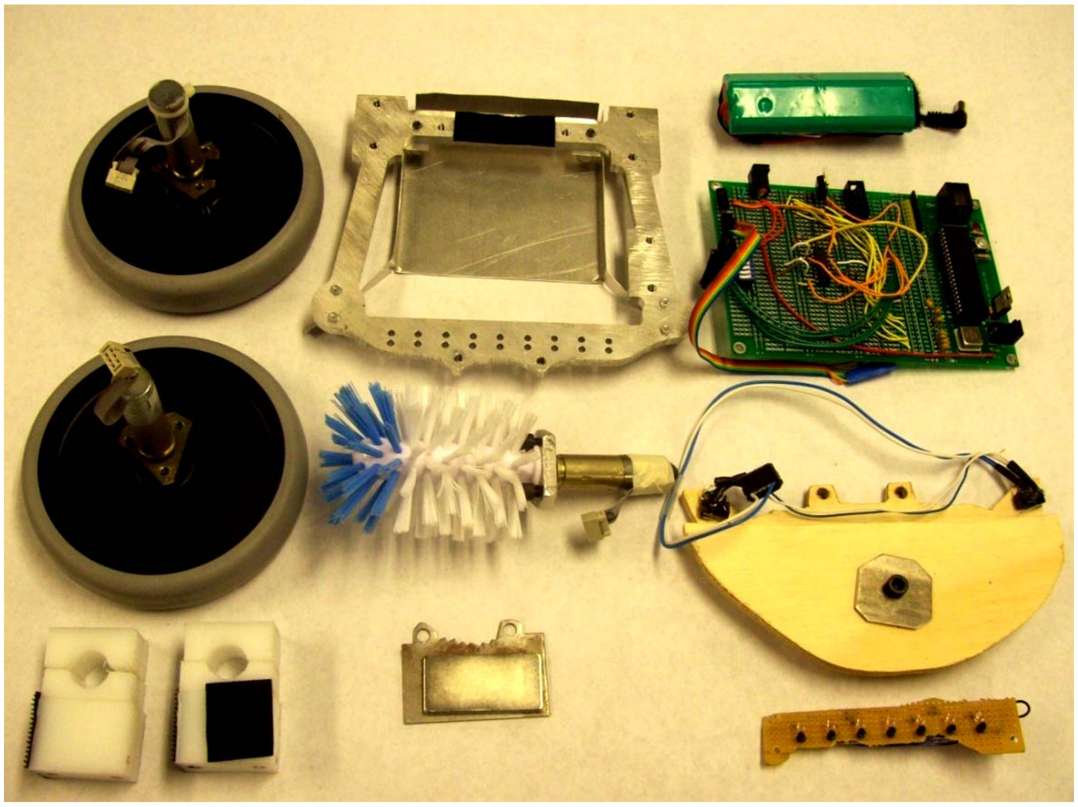
The circuit below is used to detect objects directly in front of the sensor. The potentiometer can be used to adjust for ambient lighting conditions. The output is an analog signal like the one at the top of the page - dependent on distance. The sensor works best when it is at least partially-shielded from ambient light. This output is then connected to the analog input pins of the PIC to read the sensor values.



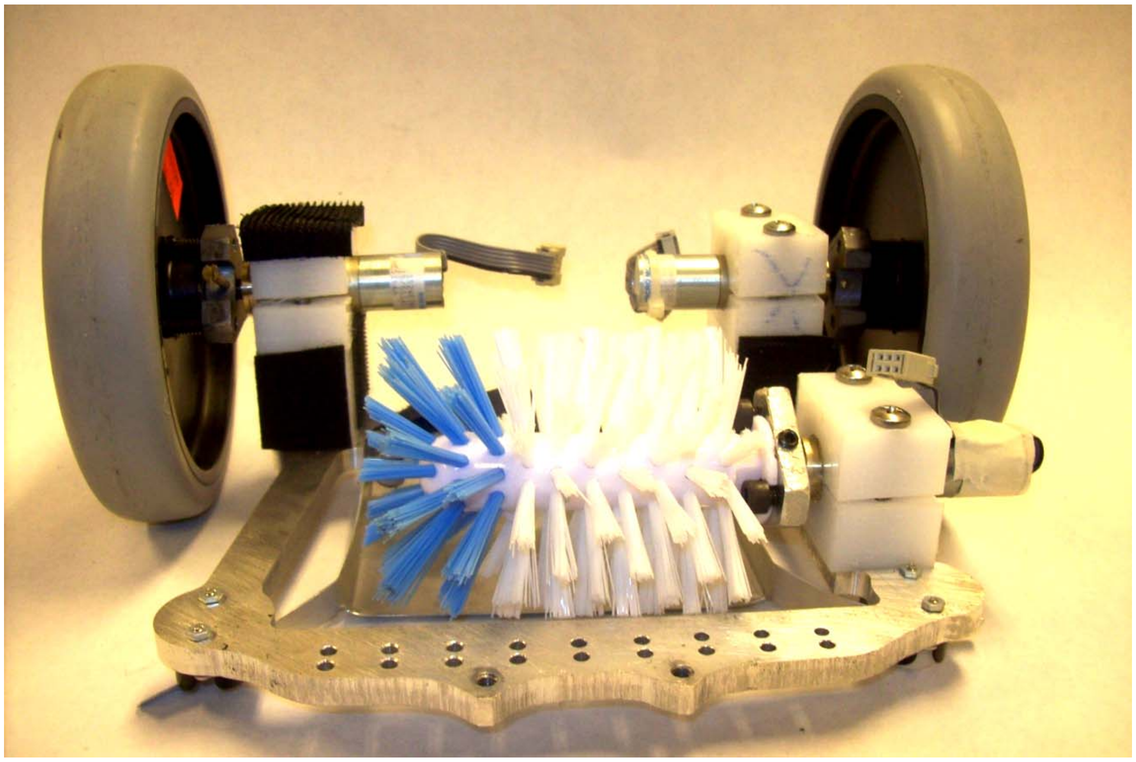
Appendix D: Photographic Representation of the Robot Assembly

The following images show a step-by-step representation of the assembly of our robot.

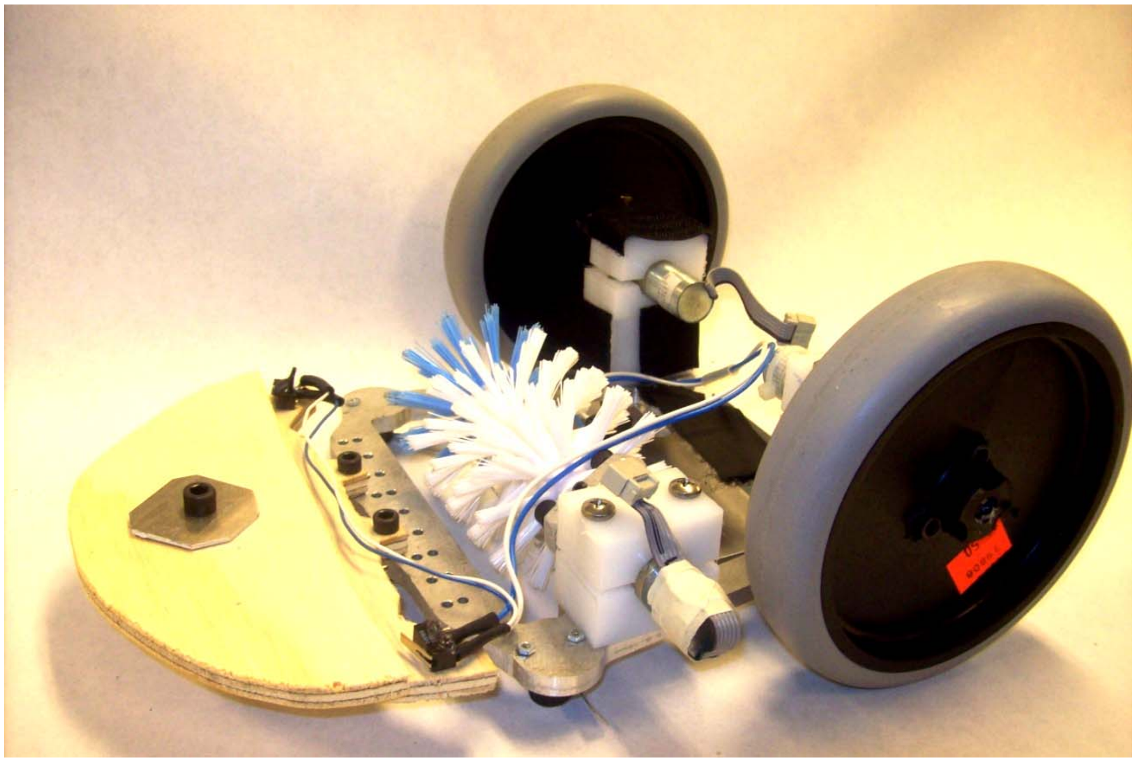
1. This shows the components required for assembly:



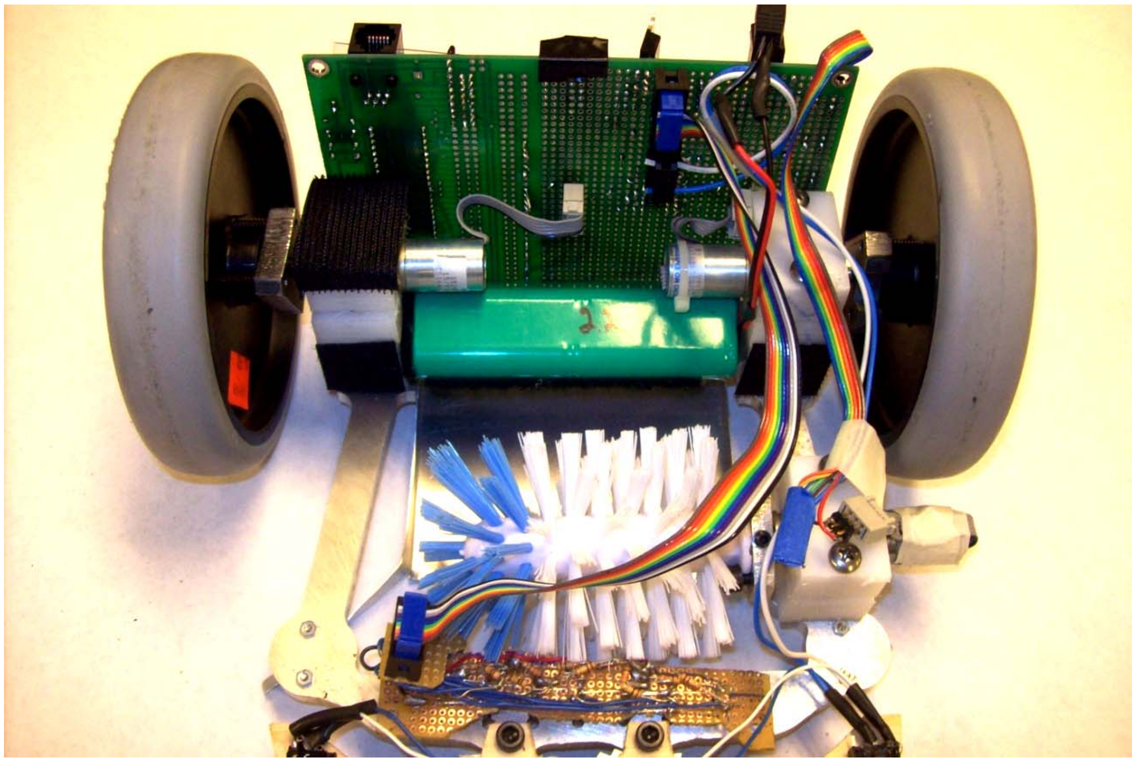
2. Second, we fix the wheel assemblies, motors, the token collector on the base board:



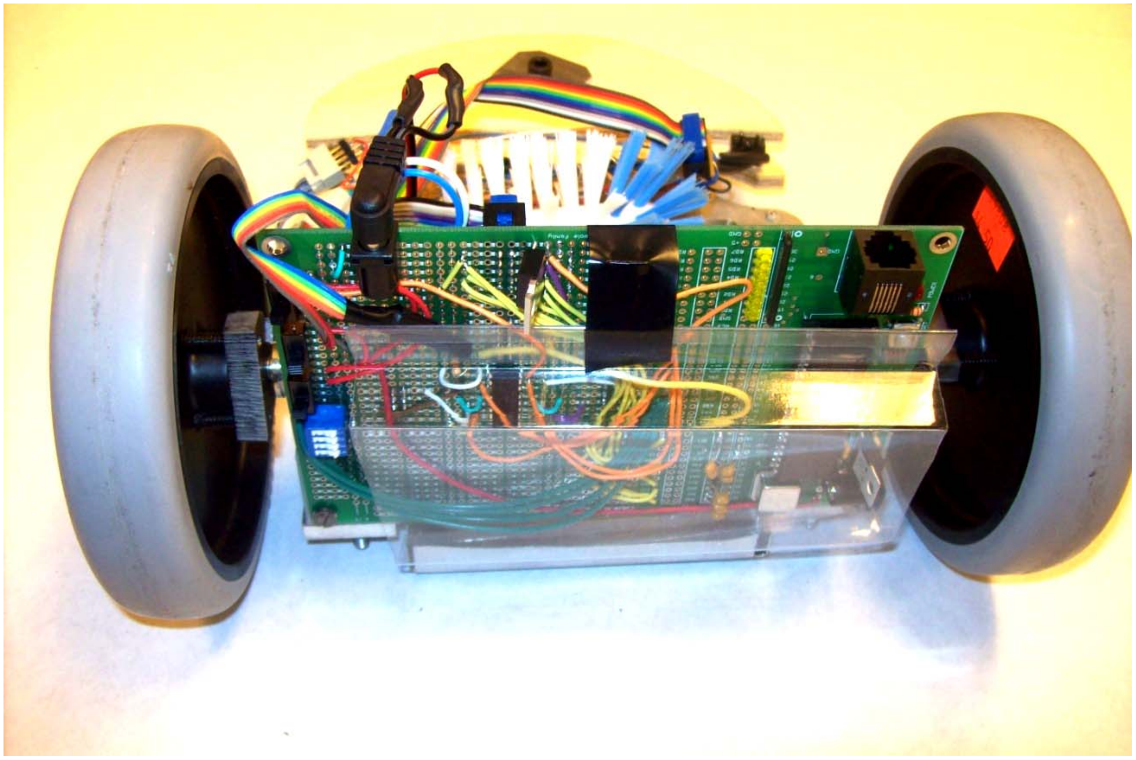
3. Next, we attach the bump sensor:



4. Then we attach the optical sensors, controller circuitry and the battery.



5. As a final safety precaution, we add a plastic guard on the back of our circuit board to prevent other robots from damaging it by hitting from the back.



Appendix E: PIC-C Source Code

Configuration Header File

```
/***************************************fork.h******************************/
/*
ROBOT CONFIGURATION HEADER FILE
Sourya Roy/Nick Renold/Edward Jen - 05/2010
*/
#ifndef FORK_H
#define FORK_H

//GLOBAL SETTINGS
#define SPEED 64
#define LOSPEED 35

//ADC Sensor Subtractions
#define R3_SUB 25
#define R2_SUB 25
#define R1_SUB 25
#define C_SUB 25
#define L1_SUB 20
#define L2_SUB 25
#define L3_SUB 40

//ADC Pin List
#define RIGHT3 0
#define RIGHT2 1
#define RIGHT1 2 //changed
#define CENTER 3 //changed
#define LEFT1 4
#define LEFT2 5
#define LEFT3 7

//Sensor LED Enable Pin
#define SENSORS PIN_C0

//PWM
#define motor1PWMdefault 40  //default PWM value for motor 1 (CCP1 - Right)
#define motor2PWMdefault 40  //default PWM value for motor 2 (CCP2 - Left)
#define Upperlimit SPEED
#define Lowerlimit 0
#define tPos 0  //target position (DO NOT CHANGE)
#define DELAYTIME 5  //delay time for sensor measurements

//FUNCTIONS
//twodrive.c - PWM Motor Control
void setup_motors(); //initialize PWM
void PWMSET(signed int,int); //PWM sentup function (internal)
void rotate(int); //rotate
void forward(int); //go forward or backward
void stop (); //stop motors

//sensors.c - Sensor ADC Output
void setup_sensors(); // Setup ADC sensor input
int read_sensor(int); // Reads ADC and outputs value
void calib_sensors(); //caliberate sensors
void calib_sensors_flash(); //calibrate pulsed sensors
void read_sensors(int *); // Reads sensors into array of 7 ints
void read_pulsed_sensors(int *);

#endif
```

Main Program Routine

```
/***************************************main.c******************************/
/*
MAIN CONTROL PROGRAM
Sourya Roy/Nick Renold/Edward Jen - 05/2010
*/
#include <18f4520.h>
#include "fork.h"

#fuses HS,NOLVP,NOWDT,NOPROTECT,NOWRTD
#use delay(clock=40000000)

//sensors
int r3_in,r2_in,r1_in,c_in,l1_in,l2_in,l3_in;
int1 r3,r2,r1,c,l1,l2,l3;

//PID
float mPos = 0;  //current position

//bunch of random variables needed to carry out PID calculation LINE SENSOR PART
float Prop = 0;
float Temp = 0;
float Integral = 0;
float Diff = 0;
float Difflast = 0;
float Control = 0;
float Control_prev = 0;
float Rate = 0;
float Deriv = 0;


float Kp = 8.0;  //Proportionality Constant (P)
float Kd = 1.5; //Integration Constant (I)
float Ki = 0.0;  //Derivative Constant (D)

int isFirstLoop = 1;

int RIGHT3_TH=0,RIGHT2_TH=0,RIGHT1_TH=0,CENTER_TH=0,LEFT1_TH=0,LEFT2_TH=0,LEFT3_TH=0;

int flag=0; //flag for bump-rotate


void main()
{


   //initialize sensors
   setup_sensors();
   //initialize motors
   setup_motors();
   stop();
   //switch on sensors
   output_high(SENSORS);


   //[OPTION] calibrate sensors
   if (input(PIN_B0)==1) {

      output_high(PIN_D7);

      delay_ms(1000);
      calib_sensors();
   output_d(255); //success

   while(1);
}

output_d(255);

//load sensor data
RIGHT3_TH=read_eeprom(0);
RIGHT2_TH=read_eeprom(1);
RIGHT1_TH=read_eeprom(2);
CENTER_TH=read_eeprom(3);
LEFT1_TH=read_eeprom(4);
LEFT2_TH=read_eeprom(5);
LEFT3_TH=read_eeprom(6);

PWMSET(SPEED-4,1);
PWMSET(SPEED,2);
delay_ms(2000);
output_d(0);
output_high(PIN_D7);

//[OPTION] initialize slo-mo
if (input(PIN_B1)==1) {
   output_low(PIN_D7);
}


while(1)
 {

    r3_in=read_sensor(RIGHT3);
    r2_in=read_sensor(RIGHT2);
    r1_in=read_sensor(RIGHT1);
    c_in=read_sensor(CENTER);
    l1_in=read_sensor(LEFT1);
    l2_in=read_sensor(LEFT2);
    l3_in=read_sensor(LEFT3);

    //DEBUG
    if (r3_in>RIGHT3_TH) {
        output_high(PIN_D0);
        r3=1;
    }
    else {
        output_low(PIN_D0);
        r3=0;
    }

   if (r2_in>RIGHT2_TH) {
        output_high(PIN_D1);
        r2=1;
    }
    else {
        output_low(PIN_D1);
        r2=0;
    }

    if (r1_in>RIGHT1_TH) {
        output_high(PIN_D2);
        r1=1;
    }
    else {
        output_low(PIN_D2);
        r1=0;
    }

    if (c_in>CENTER_TH) {
        output_high(PIN_D3);
        c=1;
    }
    else {
        output_low(PIN_D3);
        c=0;
    }

   if (l1_in>LEFT1_TH) {
        output_high(PIN_D4);
        l1=1;
   }
    else {
        output_low(PIN_D4);
        l1=0;
    }

    if (l2_in>LEFT2_TH) {
        output_high(PIN_D5);
        l2=1;
   }
    else {
        output_low(PIN_D5);
        l2=0;
    }

   if (l3_in>LEFT3_TH) {
        output_high(PIN_D6);
        l3=1;
   }
    else {
        output_low(PIN_D6);
        l3=0;
    }

    //NEW PID
    //0000001 = 6
    //0000011 = 5
    //0000010 = 4   T U R N
    //0000110 = 3
    //0000100 = 2 L E F T
    //0001100 = 1
    //0001000 = 0 ----------
    //0011000 = -1
    //0010000 = -2  T U R N
    //0110000 = -3
    //0100000 = -4  R I G H T
    //1100000 = -5
    //1000000 = -6
    //0000000 = 7 or -7 depending on previous values

   if (r1==0 && r2==0 && r3==0 && c==0 && l1==0 && l2==0 && l3==1)
      mPos=7;
   else if (r1==0 && r2==0 && r3==0 && c==0 && l1==0 && l2==1 && l3==1)
      mPos=6;
   else if (r1==0 && r2==0 && r3==0 && c==0 && l1==0 && l2==1 && l3==0)
      mPos=5;
   else if (r1==0 && r2==0 && r3==0 && c==0 && l1==1 && l2==1 && l3==0)
      mPos=3;
   else if (r1==0 && r2==0 && r3==0 && c==0 && l1==1 && l2==0 && l3==0)
```

```
            mPos=2;
      else if (r1==0 && r2==0 && r3==0 && c==1 && l1==1 && l2==0 && l3==0)
            mPos=1;
      else if (r1==0 && r2==0 && r3==0 && c==1 && l1==0 && l2==0 && l3==0)
            mPos=0;
      else if (r1==0 && r2==0 && r3==1 && c==1 && l1==0 && l2==0 && l3==0)
            mPos=-1;
      else if (r1==0 && r2==0 && r3==1 && c==0 && l1==0 && l2==0 && l3==0)
            mPos=-2;
      else if (r1==0 && r2==1 && r3==1 && c==0 && l1==0 && l2==0 && l3==0)
            mPos=-3;
      else if (r1==0 && r2==1 && r3==0 && c==0 && l1==0 && l2==0 && l3==0)
            mPos=-4;
      else if (r1==1 && r2==1 && r3==0 && c==0 && l1==0 && l2==0 && l3==0)
            mPos=-5;
      else if (r1==1 && r2==0 && r3==0 && c==0 && l1==0 && l2==0 && l3==0)
            mPos=-6;
      else if (mPos >= 0.0)
                  mPos=8;
      else if (mPos <= 0.0)
                  mPos=-8;


        //[OPTION] Slo-Mo BANG BANG Approach
        if (input(PIN_B1) == 1) {
            if(c==1) //go straight
                  forward(LOSPEED);
            else if ((r1+r2+r3)>(l1+l2+l3)) { //turn right
                  PWMSET(-SPEED,1);
            }
            else { //turn left
                  PWMSET(SPEED,1);
            }

        }


      Diff = tPos - mPos;  //P
      Prop = Diff * Kp;

      Integral += Diff; // I
      Integral *= Ki;

      Rate = Diff - Difflast; // D
      Deriv = Rate * Kd;

      Difflast = Diff;

      Control = Prop + Deriv + Integral;

      if (isFirstLoop == 1){
            isFirstLoop = 0;
            Control_prev = Control + 1;  //dummy value so that Control isn't equal to
Control_prev for FIRST RUN
      }

      if (input(PIN_B5) == 1){  //bump sensor is activated; both sensors are depressed
-> wait three seconds and back up
            PWMSET(SPEED,1);
            PWMSET(SPEED,2);
            delay_ms(1500);
            if (input(PIN_B5) == 1){
                  PWMSET(-SPEED,1);
                  PWMSET(-SPEED,2);
```

```
                delay_ms(1500);
                if (flag==0) {
                    rotate(SPEED);
                    delay_ms(1500);
                    stop();
                    forward(SPEED);
                    delay_ms(1000);
                    mPos = 0;
                    flag=1;
                }
                else {
                   rotate(SPEED);
                   delay_ms(1000);
                   stop();
                   forward(SPEED);
                   delay_ms(1500);
                   mPos = 0;
                   flag=0;
                }
             }


            }

      if (Control != Control_prev){ //change PWM ONLY if needed, else skip

             if (input(PIN_B1) != 1) { //follow PID

               if (Control < 0){ //Turn left
                   PWMSET(motor2PWMdefault + Control,2);
                   PWMSET(motor1PWMdefault,1);
               }
               else {//Turn right
                   PWMSET(motor1PWMdefault - Control,1);
                   PWMSET(motor2PWMdefault,2);
               }
            }

            Control = Control_prev;


            if (input(PIN_B1) == 1) {
                  if (Control < 0){ //Turn left
                PWMSET(30 + Control,2);
                PWMSET(30,1);
                }
                else {//Turn right
                    PWMSET(30 - Control,1);
                    PWMSET(30,2);
                }
            }


       }
         //delay_ms(DELAYTIME);
         }
}
```

PWM Motor Driver

```
/***************************************twodrive.c******************************/
/*
2 WHEEL PWM MOTOR DRIVER
Sourya Roy/Nick Renold/Edward Jen - 05/2010
*/
#include <18f4520.h>
#include "fork.h"

#fuses HS,NOLVP,NOWDT,NOPROTECT
#use delay(clock=20000000)

//motor polarities
#define M1p  1
#define M2p  1 //polarity for motor 2 corrected in H/W

//for motor PWM
#define M1PWM(num) set_pwm1_duty(num) //c2
#define M1REV(bit) OUTPUT_BIT(PIN_C5,bit)  //c5
#define M2PWM(num) set_pwm2_duty(num) //c1
#define M2REV(bit) OUTPUT_BIT(PIN_C4,bit)  //c4

void setup_motors()     //initialize PWM
{
   //setup_timer_2(T2_DIV_BY_1, 255, 16);
   setup_timer_2(T2_DIV_BY_16, SPEED, 16);
   //setup_timer_2(T2_DIV_BY_4, 63, 16); // 5Mz instruction cycle, pwm cycle every
4*64 cycles (19.5khz)
                                     // interrupt every 16th of that (not used)

   setup_ccp1(CCP_PWM);                // PWM output on CCP1/RC2, pin 17
   setup_ccp2(CCP_PWM);                // PWM output on CCP2/RC1, pin 16.  PWM2 can be
moved to pin 36; see ServoSkeleton
}

void PWMSET(signed int pwm,int motor){
 int cycle;
 int1 rev_bit;
   if(pwm<0){
    rev_bit=1;
    cycle=pwm+SPEED;//cycle=pwm+64;
 }
 else{
    rev_bit=0;
    cycle=pwm;
 }
   if(motor==1){
    M1PWM(cycle);
    M1REV(rev_bit);
 }
 else if(motor==2){
    M2PWM(cycle);
    M2REV(rev_bit);
 }
}

void rotate(int i) {       //rotate; i -> speed (-64 to +64)
//rotates leftwise

    PWMSET(M1p*i,1);       //motor 1 moves in (+)ve direction
    PWMSET(-M2p*i,2);       //motor 2 moves in (-)ve direction
    //output_high(PIN_D1);
}

void forward(int i) {       //move straight forward; i -> speed (-64 to +64)

    PWMSET(M1p*i,1);       //both motors move straight in (+)ve direction
    PWMSET(M2p*i,2);
    //output_high(PIN_D0);
}

void stop () {       //stops all motion
   PWMSET(0,1);
   PWMSET(0,2);
   //output_low(PIN_D0);
   //output_low(PIN_D1);
}
```

Sensor Input Driver

```
/**************************************sensors.c******************************/
/*
SENSOR INPUT DRIVER
Sourya Roy/Nick Renold/Edward Jen - 05/2010
*/
#include <18f4520.h>
#include "fork.h"

#fuses HS,NOLVP,NOWDT,NOPROTECT
#use delay(clock=20000000)

//#DEVICE ADC=8                   // set ADC to 10 bit accuracy, or it could be just 8


// Setup ADC sensor input
void setup_sensors() {

   setup_adc_ports(AN0_TO_AN7);        // Enable analog inputs; choices run from just
AN0, up to AN0_TO_AN11
   setup_adc(ADC_CLOCK_INTERNAL);      // the range selected has to start with AN0
   //setup_adc_ports(AN0_TO_AN6|VSS_VDD);
   //setup_adc(ADC_CLOCK_DIV_2|ADC_TAD_MUL_0);

}



// Reads ADC and outputs value
int read_sensor(int s) {

      int value;

      set_adc_channel(s);      // there's only one ADC so select which input to connect
to it; here pin AN0
      delay_us(10);            // wait 10uS for ADC to settle to a newly selected input
      value = read_adc();      // now you can read ADC as frequently as you like

      return value;

}


void calib_sensors() { //sensor auto calibration
   int r3=0,r2=0,r1=0,c=0,l1=0,l2=0,l3=0,temp=0;
   int16 counter;

   output_high(PIN_D6);

      for (counter=0; counter<20000; counter ++) {

        //rotate
        rotate(25);

        //find max sensor readings
         temp=read_sensor (RIGHT3);
         if (temp>r3)
            r3=temp;
         temp=read_sensor (RIGHT2);
         if (temp>r2)
            r2=temp;
         temp=read_sensor (RIGHT1);
         if (temp>r1)
            r1=temp;
         temp=read_sensor (CENTER);
         if (temp>c)
            c=temp;
         temp=read_sensor (LEFT1);
         if (temp>l1)
            l1=temp;
         temp=read_sensor (LEFT2);
         if (temp>l2)
            l2=temp;
         temp=read_sensor (LEFT3);
         if (temp>l3)
            l3=temp;
      }

   stop(); //stop motors

   //decrement max values by subtraction constant (SUB)
   r3-=R3_SUB;
   r2-=R2_SUB;
   r1-=R1_SUB;
   c-=C_SUB;
   l1-=L1_SUB;
   l2-=L2_SUB;
   l3-=L3_SUB;

   //write to EEPROM
   write_eeprom(0,r3); //keep eeprom address numbers same as adc ports (0,1,2 etc)
   write_eeprom(1,r2);
   write_eeprom(2,r1);
   write_eeprom(3,c);
   write_eeprom(4,l1);
   write_eeprom(5,l2);
   write_eeprom(6,l3);

  output_low(PIN_D6);
}
```

References

1. Mechatronics Wiki (http://hades.mech.northwestern.edu/wiki/)
2. Society of Robots (http://www.societyofrobots.com)
3. Hobby Engineering – Robotics (http://www.hobbyengineering.com/DeptRB.html)
4. Datasheets of the components used.
5. Robot Room by David Cook (http://robotroom.com/)
6. Cook, David. *Intermediate Robot Building*. Berkeley, CA: Apress, 2004. Print.
7. Wright, Eddy. "PID for Line Following." *Chibots*. 27 Oct. 2007. Web. 6 May 2010. <http://www.chibots.org/index.php?q=node/339>.