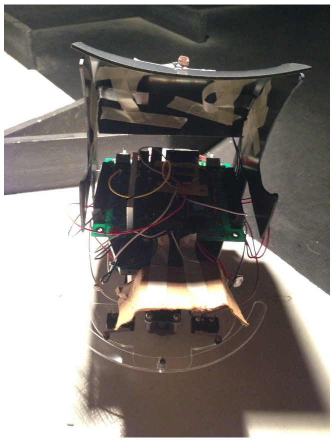
Robot Design Project



December 3rd 2013, E 121

Section H, Group number 4
Rong Lei,
Edward Carannante,
Kelly Jardine

“I pledge my honor that I have abided by the Stevens Honor System”

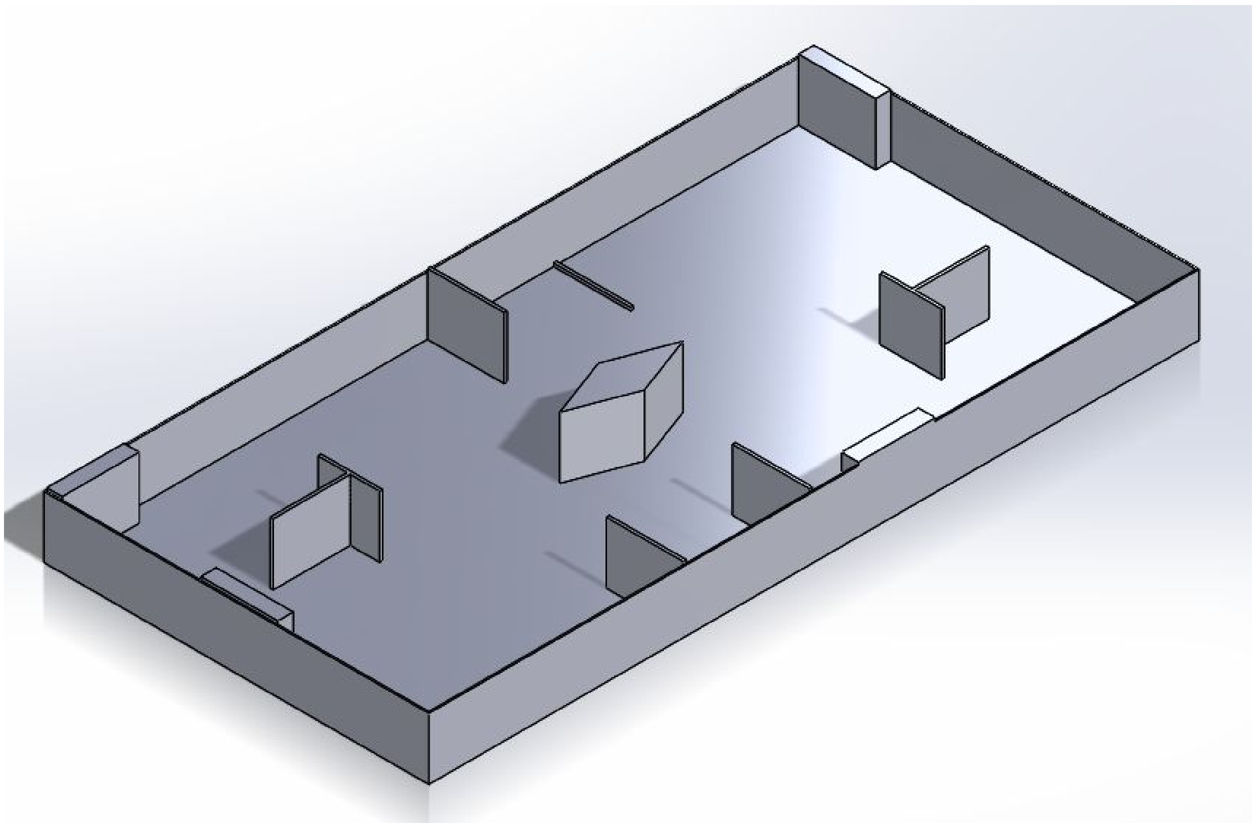
Abstract

The assignment was to design a robot that could navigate the course quickly and efficiently and to find two light targets. We completed this task by designing the best possible programs and design to create the most functional robot. Our bumpers effectively can tell which side has been struck by an object, and will react by turning in the appropriate direction. We positioned the light sensors in a specific manner as to accurately detect floor location/color, the target light locations and the overhead light. This process was made possible by staying organized, proper planning, and time management.

Introduction

In this report the group will explain the breakdown of the robot completion, which includes but is not limited to: the rules of competition, the requirements of the robot, the build of the robot and reasoning for deciding to build the robot in this manner, and the overall success of the final project. In the planning and construction of this robot, groups learned the basic steps in project development, as well as valuable team working skills. The robot that the team has constructed for this project must meet certain requirements, which include navigating through a course filled with obstacles (shown below) in order to reach a light which is on enemy ground and return to its own home ground. The competition is scored based on the robots efficiency and its overall design and functionality. As demonstrated on the attached Gantt chart, the tasks were divided amongst the team members. One group member was primarily responsible for the programming of the robot, and the other two members were assigned to the physical aesthetical build of the robot. The use of a Gantt chart was helpful in keeping the team on track timewise.

3-D Model of the Arena



Requirements:

The only reason to build a robot is to successfully complete a challenge or task. Our objective was to build a robot that can successfully maneuver around a course and turn off two target lights faster than an enemy robot. Our base was standard. You are not allowed to alter anything in the core robot. This includes the PVC mounting platforms, the microprocessor, the gear box, motors and wheels.

In order to put effective programs in effect to allow our robot to move, we had to construct a **Microprocessor Board**.

The course is half black and white, allowing our group to design a program and a **Floor Sensor Module.** Every group receives parts for it. The Floor Sensor Module will determine what side of the course we are on and determine when we have crossed to the other side. This helps distinguish which light is the target light, rather than the base light. In order to create this we had to construct a circuit for the LED's.

Our group also had to construct a **Bumper Interrupt System** because there are a few obstacles that the robot must maneuver around. The group created a design on solid works that will do this. Our group was given interrupt switchers that we could mount on plastic to make the bumper interrupt system. This determines if a bumper is hit or not. The group built a circuit for this. A requirement our group made was to make the bumpers as simple as possible and to be no wider than half an inch with respect to the robot's base.

The group is provided with **light sensors** which we must position effectively in order to get consistent readings. This will allow us to efficiently locate the target lights so we can drive in the right direction and turn off the target light.

Lastly the **starting location** and the position of **the target lights** will be unknown. This requires that our group design a program that can tell which side we start on, using the **floor sensor module.** After that the robot must navigate to the other side using the light sensors then turn off the target light. Finally, the robots will then turn around and find the target line on the starting side.

System Design

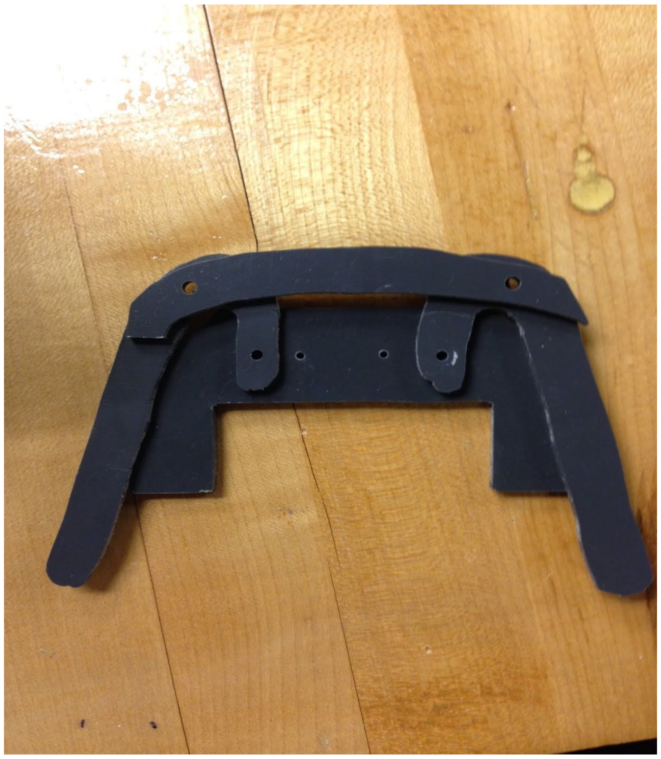
In order to create the most successful robot possible, the group needed to create a “win strategy.” For the group, this strategy consisted of developing and building functional bumpers as well as a program that operates the bumpers. The strategy also consisted of developing a unique system of light sensors. The unique light sensor setup that the group has used consisted of one floor sensor, one top light sensor, and three target sensors, one on the left, right, and in front. The group chose to utilize an extra target sensor in attempt to build a more efficient robot, ideally this extra sensor would help the robot to locate and navigate to the light quicker than if it had two sensors. The group’s bumper system consisted of one semicircle piece which protected the front of the robot. When the semicircle was struck by an object, the bumper would rotate and strike a sensor on the robot and the robot would rotate in the opposite direction. A unique feature of the bumper system is that if the robot would hit an object head on, activating both bumper sensors simultaneously the robot would turn in the direction that the target light is the strongest. The team had originally designed a bumper system that consisted of 3 pieces, one on the left, one on the right, and a third in the front, connecting them to each other. This design failed to be functional for the robot because both bumper sensors could not be activated simultaneously, so the group then had to redesign a new bumper system. * possibly add mirrors *

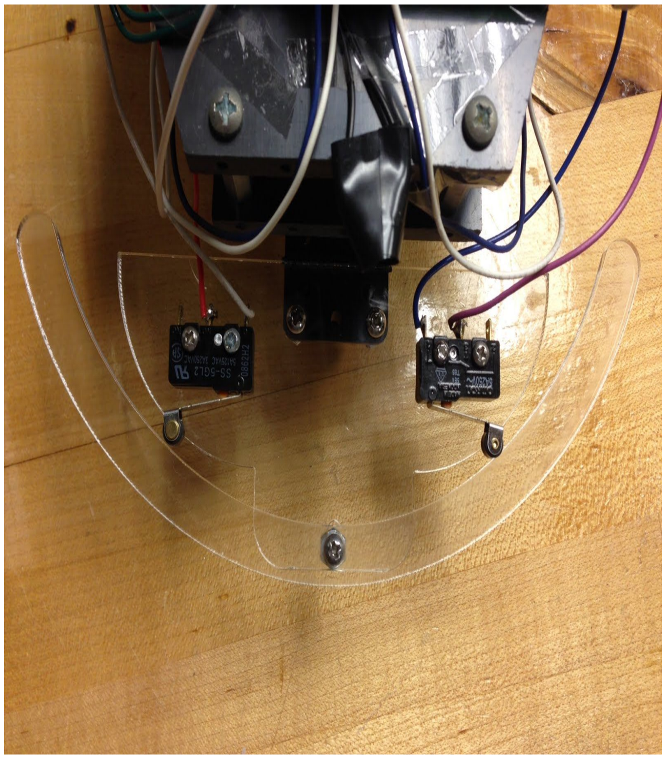
Pugh Matrix for target light sensor suite

Mechanical Design

How was and why did you mount your components as you did?
How was the floor sensor module constructed and mounted?

When we first talked about how we were going to make our bumpers, we all thought it would be best to make a three part bumper system like the picture on the left. After testing it out and adjusting it, there was simply no way the design would work, so we scratched it a designed a new single bumper bumper system like the picture on the right. It only took about an extra couple hours to complete the new bumpers between creating it on solidworks, having it cut out at the laser cutter, and screwing the screws in.





(Old Bumper Design) **(New Bumper Design)**

Our Light sensors had to be shielded in order to prevent light from the target light being picked up by the over head light sensors and the front light sensors to prevent light from the overhead light being picked up. To avoid this issue, we positioned the overhead light sensors on the back of our shark shell. This guarantees that no light from the target light will be picked up by the overhead light sensors.
The team produced a logo in Solid Works and printed the logo utilizing the laser cutter. This

logo was a lifeguard symbol that was mounted in the sharks mouth.

Electrical and Wiring design

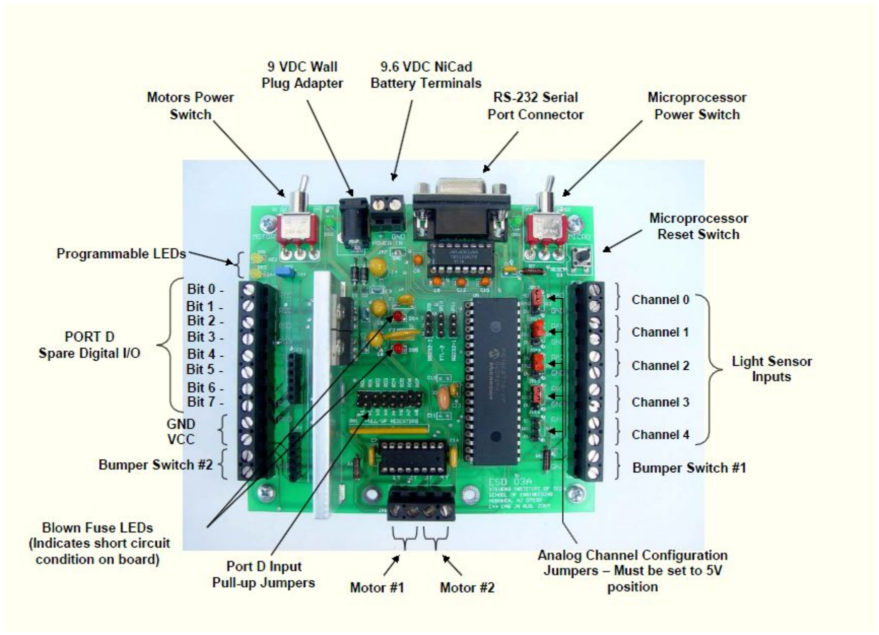
1. PIC Board

PIC microcontroller circuit board was a kind of assembled circuit board based on PIC16F877A microprocessor. The main chip stored the program downloaded from computer, and the program would remain in the chip until another program overwrote it.

The team used the RS-232 serial cable with a serial to USB converter to connect the RS-232 Serial Port Connector on the PIC board and the USB Port on a laptop.

The step to connect and download the program is:

(1)Select 19200 Baud and select COM4 port in the "Bootloader Window" in software named MPLAB IDE.
(2)Enter the path of the program in a ".Hex" file located in the "Release" folder of project.
(3)Turn on the "MICRO" power switch on the PIC board.
(4)Depress and hold down the PIC microcontroller board reset button.
(5)Click the "Write Flash" box on the laptop.
(6)Release the reset button.
(7)After a successful download, close the "Bootloader Window".
Below is a picture of PIC microcontroller board.


9 VDC Wall Plug Adapter
9.6 VDC NiCad Battery Terminals
Motors Power Switch
RS-232 Serial Port Connector
Microprocessor Power Switch
Microprocessor Reset Switch
Programmable LEDs
PORT D Spare Digital I/O
Bit 0 -
Bit 1 -
Bit 2 -
Bit 3 -
Bit 4 -
Bit 5 -
Bit 6 -
Bit 7 -
GND
VCC
Bumper Switch #2
Channel 0
Channel 1
Channel 2
Channel 3
Channel 4
Light Sensor Inputs
Bumper Switch #1
Blown Fuse LEDs (Indicates short circuit condition on board)
Port D Input Pull-up Jumpers
Motor #1
Motor #2
Analog Channel Configuration Jumpers – Must be set to 5V position

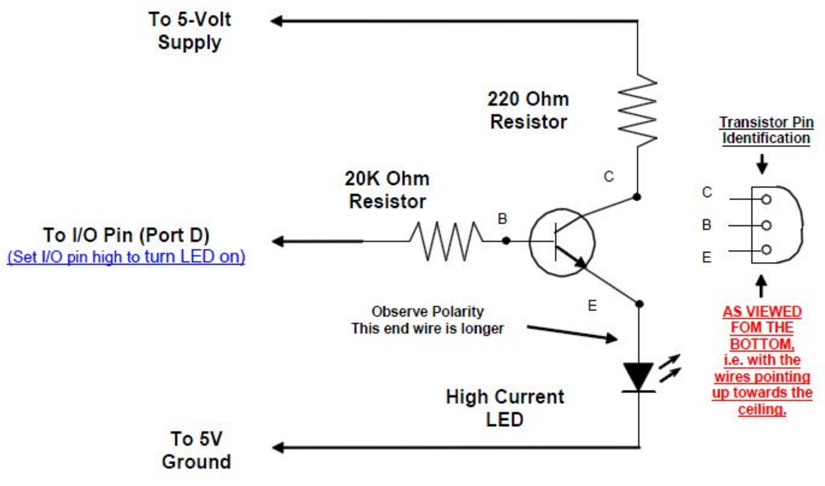
2. Electrical Design

(1)FSM (Floor Sensor Module)

a) Electrical Design

The Floor Sensor Module was used for detecting the color on the arena floor. Thus, the robot would know on which side it was, and then robot could decide to navigate or target.

The group assembled an additional electronic interfacing circuitry to connect the LED.A small printed circuit board was used to hold every component including a light sensor, a high current Light Emitting Diode (LED), a transistor and two resistors. This circuitry was considered to supply the large amount of current needed to light the LED, which could not be directly supplied by the PIC chip I/O pin. A transistor was used to switch a 5 volt source to a control signal provided by the PIC microcontroller in order to protect the LED from burning out.

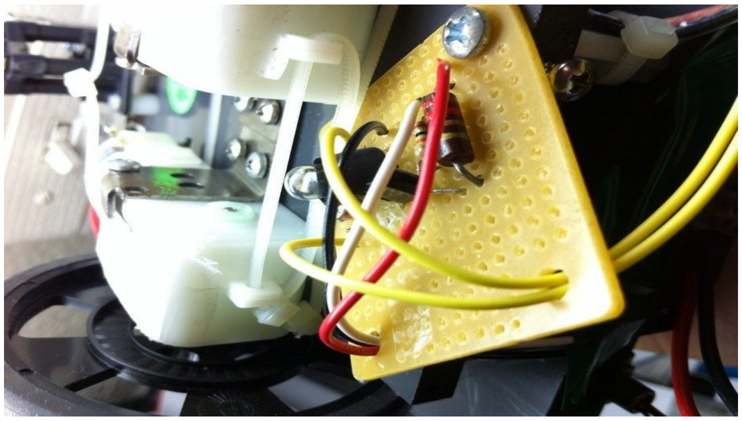

To 5-Volt Supply
220 Ohm Resistor
Transistor Pin Identification
20K Ohm Resistor
C
B
E
To I/O Pin (Port D)
(Set I/O pin high to turn LED on)
Observe Polarity
This end wire is longer
AS VIEWED FOM THE BOTTOM, i.e. with the wires pointing up towards the ceiling.
High Current LED
To 5V Ground

b) Unique Considerations

i. The team used a black tape attached around the LED light to constrain the light range for the light sensor. Thus, the distribution of the light reflected from white floor and black floor had a distinguish difference.

ii. The team soldered the end of components in linear pattern, so that seeing from the back of circuit board, it was easier to check the function of total circuit.

iii. The team arranged the wires all through the holes on the circuit board to constrain the room they needed and in case that they tied together.



c) Wire Color Coding

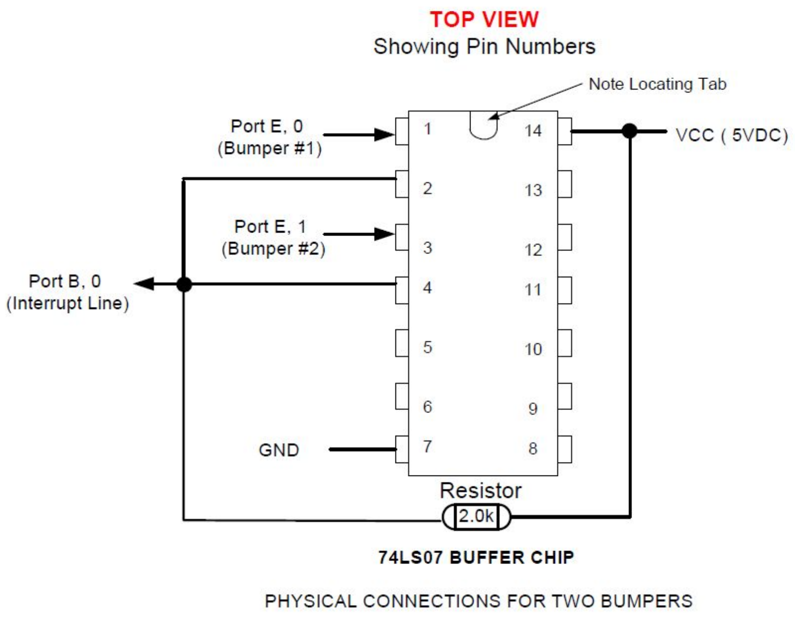
(2) BIS (Bumper Interrupt Support)

a) Electrical Design

The Bumper Interrupt Support was designed to make quick respond to the signal from the buffers and get away from the obstacles in the way. The main chip used in this circuit was a 14 pin dual in-line chip HD74LSO7P.

The type of circuit foundation was decided by the exact number of bumpers that robot would use, ranging from one to six, which was depended by the total numbers of separate buffer circuits that the 14 pin dual in-line chip contained. In consideration of lighting the entire weight and making robot more sensitive, the team finally picked two-bumpers system. The group members assembled all components including 5 wires, a 2.0k resistor and a buffer chip socket on another small printed circuit board.

Below was a circuit for the two-bumpers BIS.



b) Unique Considerations
   i. The team used electric tape to tape behind the printed circuit board to prevent a short.

c) Wire Color Coding

Software Design and Coding

1. Software Process

a) Analyze the problem, figure out what functions a robot needs, what is the final goal.
b) Build logical structure for the robot, analyze its possibility.
c) Design flowchart for each function, transfer logical structure into concrete graphs.
d) Record data in the real field for some specific function, such as navigation and target part, the team recorded the light sensor data to determine the distribution lines for different field color.
e) Write code for each function in C language, debug and build the file.
f) Download the program in the chip and test robot in real field, find problems and fix it several times.

2. Software Tools

a) MPLAB X: used for editing the C source code and integrating the additional tools.
b) HI-TECH Compiler: transfer the C source code into assembly code that specially fitted

PIC controller.

c) HI-TECH Assembler: Assembles the assembly code into a.HEX file, which is machine-readable, that can download into the flash memory of the PIC chip.

d) TinyBootloader: sends the.HEX file, through the USB connection, to the PIC board.

e) TinyBootloader Terminal: allows the viewing of data sent from the PIC board for diagnosing.

3. Software Requirements

4. Subroutines

a) Pre-defined Subroutines

1.) void configurePIC ( ); (Only called once from main() at top of program)

Purpose: Configures the PIC to its baseline E121 design project configuration. This includes setting all I/O ports/pins to be digital inputs, except for the five light sensor inputs which are set for analog operation.

2.) void motorspeed (modernism, speed);

Purpose: Individually sets the speed of the requested motor. This function operates by sending a pulse width modulated signal to the motor in the range of 30 to 100% duty cycle. At 100% the motor will turn full speed. At 30 % the motor will turn much slower according to the load experienced, i.e., weight of a robot.

3.) void output_high (port, bit);

Purpose: Sets the specified I/O pin to be an output pin and initializes it to be high (logic level 1). This put 5 volts on to the output pin. Input Parameters: port = ‘A’, ‘B’, ‘C’, ‘D’, ‘E’ (lower case is also accepted).

4.) void output_low (port, bit);

Purpose: Sets the specified I/O pin to be an output pin and initializes it to be low (logic level 0). This grounds the output pin.

5.) unsigned int read_adc (channel);

Purpose: Reads the voltage present at the specified analog input channel and returns that value back to the calling function. This function was used to read the output of the light sensors.

6.) char read_input (port, bit);

Purpose: Used to read digital inputs. Returns the digital logic level (0 or 1) present at an input pin 0 indicates the pin is connected to ground; pin 1 indicates the pin is connected to 5 VDC.

7.) void putdata (data);

Purpose: Used to send data to a terminal program so it can be viewed.

8.) void putchar (data);

Purpose: Sends the specified ASCII character over the RS-232 serial port or USB port.Used to send a character to a terminal program so it can be viewed.

9.) void pause (data);

Purpose: To introduce a delay in the program on the order of milliseconds.

b) Unique Subroutines

1) Void motorsoff(void)

i. Explanation

This function was used for shutting off the switches of motors immediately, a pre-defined function output_low() was activated for turning off each bit 0,5,3,4 on port C.

ii. Algorithm

```
void motorsoff()
{
 output_low('C',0);
 output_low('C',5);
 output_low('C',3);
 output_low('C',4);
 return;
}
```

2) Void forward(void)

i. Explanation

The function was used to re-activate two motors and make them run clockwise seem from right side, thus, it can move forward.

ii. Algorithm

```
void forward()
{
output_low('C',5);
output_low('C',3);
 output_high('C',4);
 output_high('C',0);
 return;
}
```

iii. Unique features

Considering the different directions each bit controls, the team tested every wheel and made a chart about the exact rotate direction for each wheel.

Bit	Control wheel	Rotate Direction(Seeing from right side)
3	Left	CCW(counter clockwise)
5	Right	CCW
4	Left	CW(clockwise)
0	Right	CW

3) Void backward(void)

i. Explanation

The function was designed to make robot run backwards, this function was almost the same as void forward()function, the team just changed the bits that activate robot wheel to run counterclockwise when seeing from right side.

ii. Algorithm

```
void backward()
{
 output_low('C',0);
 output_high('C',5);
 output_low('C',4);
 output_high('C',3);
 return;
}
```

4) Void pivotleft(void)

i. Explanation

This function was used for turning left. The team used two wheels rotating simultaneously. Thus, it saved a half of time comparing to the strategy that making one wheel stand and another rotate. Bit 3 and bit 0 were activated to make both wheels rotate at correct direction.

ii. Algorithm

```
void pivotleft()
{
output_low('C',5);
output_low('C',4);
output_high('C',3);
output_high('C',0);

return;
}
```

5) Void pivotright(void)

i. Explanation

This function was used for turning left. The team used two wheels rotating simultaneously. Thus, it saved a half of time comparing to the strategy that making one wheel stand and another rotate. Bit 3 and bit 0 were activated to make both wheels rotate at correct direction.

ii. Algorithm

```
void pivotright()
{
output_low('C',0);
output_low('C',3);
output_high('C',4);
output_high('C',5);

 return;
}
```

6) Void navigation(void)

i. Explanation& Unique features

The function was designed for navigating with the help of ceiling light. When robot was standing in home side, this function would begin to work until robot found the right direction to go across the middle line of field.

Three light sensors were used when adjusting robot's direction. One of the light sensors was posted at the top side in order to locate ceiling light, other two sensors were fixed about 45 degree with respect to the centerline of robot. These two sensors were used to choose whether side to turn at beginning automatically, judging from which side the light intensity was stronger.

The programmer used two loops in this function. The first loop was to find a range of light intensity beyond a certain line (say about 20000). When the light intensity was below 20000, the robot kept rotating with full speed. When the light intensity was above that certain line, robot stopped immediately. That meant robot was faced to an approximate direction. The motors changed their speed at about half of full speed to locate ceiling light precisely.

The second loop used average sum when top light sensor was recording the light intensity data. Average sum for a short time range could minimize the random error. The robot rotated slowly and record light intensity for a short range of degree, and microchip kept comparing data. When the latest data was higher than last data, that meant the robot was pointing at appreciate direction. Then robot stopped rotating.

ii. Algorithm

```
void navigation()
{
  b=0;
sensor3 = read_adc (3);
sensor5 = read_adc (5);

sensor0[0]=read_adc(0);
sensor=sensor0[0];

while(sensor>20000)   //need test the light //no average for speed
{if(a>23)
   return;
  sensor=sensor0[0];
   if(sensor3<sensor5)
     pivotleft();
  else
     pivotright();
  pause(90);

sensor0[0]=read_adc(0);
a++;
}

motorsoff();
pause(500);
motorspeed(1,50);          //right from above
motorspeed(2,70);
 for(i=0,sum=0;i<N;i++)
{
sensor0[i]=read_adc(0);
sum+=sensor0[i];
}
average=sum/N;
sensor0[0]=average;
sensor=sensor0[0];
//need test the light
sensor3 = read_adc (3);
sensor5 = read_adc (5);

while(sensor0[0]<=sensor)         //average for precise
{
     sensor=sensor0[0];
   if(sensor3<sensor5)

   pivotleft();



   else
      pivotright();
     pause(90);
     for(i=0,sum=0;i<N;i++)
{
sensor0[i]=read_adc(0);
sum+=sensor0[i];
}
average=sum/N;
sensor0[0]=average;



}
motorsoff();
pause(500);
motorspeed(1,93);          //right from above
motorspeed(2,100);
forward();
  return;
   //test}



}
```

iii. Flowchart

7) Void target(void)

i. Explanation

ii. Unique features

iii. Algorithm

```
void target()
{
a=0;
sensor3 = read_adc (3);
sensor5 = read_adc (5);

sensor2[0]=read_adc(2);
sensor=sensor2[0];

while(sensor>20000)   //need test the light //no average for speed
{if(a>23)
    return;
   sensor=sensor2[0];
    if(sensor3<sensor5)
       pivotleft();
   else
       pivotright();
   pause(90);

sensor2[0]=read_adc(2);
a++;
}
motorsoff();
pause(500);
motorspeed(1,50);           //right from above
motorspeed(2,70);
 for(i=0,sum=0;i<N;i++)
{
sensor2[i]=read_adc(2);
sum+=sensor2[i];
}
average=sum/N;
sensor2[0]=average;
sensor=sensor2[0];
//need test the light
sensor3 = read_adc (3);
sensor5 = read_adc (5);

while(sensor2[0]<=sensor)        //average for precise
{
    sensor=sensor2[0];
  if(sensor3<sensor5)

  pivotleft();



  else
     pivotright();
    pause(90);
    for(i=0,sum=0;i<N;i++)
{
sensor2[i]=read_adc(2);
sum+=sensor2[i];
}
average=sum/N;
sensor2[0]=average;
}
motorsoff();
pause(500);
motorspeed(1,93);          //right from above
motorspeed(2,100);
forward();
  return;
  //test}



}
```

iv. Flowchart

8) Void interrupt isr(void)

i. Explanation&Unique features

This function was recommended to build a “Collision Avoidance Subsystem”.The computer interrupt was triggered when the value(voltage) on a particular input line of the microcontroller changes state, the internal hardware of the chip will immediately recognize the event, and stop whatever it was doing. Two statements “INTE=1; GTE=1” were added after “configurePIC()” to enable the interrupt function.

The team wrote the void interrupt isr() based on two-bumpers system. So the wiring coding part was pretty typtical. When either side the electrical bumper swith was triggered, robot went backward a little, then turn to the opposite side. Considering the circumstance that both bumpers were triggered simutaneously, the team made rotbot turn automatically with the help of two light sensors each was seperately fixed 45 degree with the respect to the centreline of robot. When robot was blocked by a perpendicular face sitted right in front of it and both switches were triggered, the robot turned to either side which side light intensity was tronger. The team conducted an experiment and drew a conclution that if one side had a face, that side the light sensor could get weaker light intensity data. So robot could kept turing to open area.

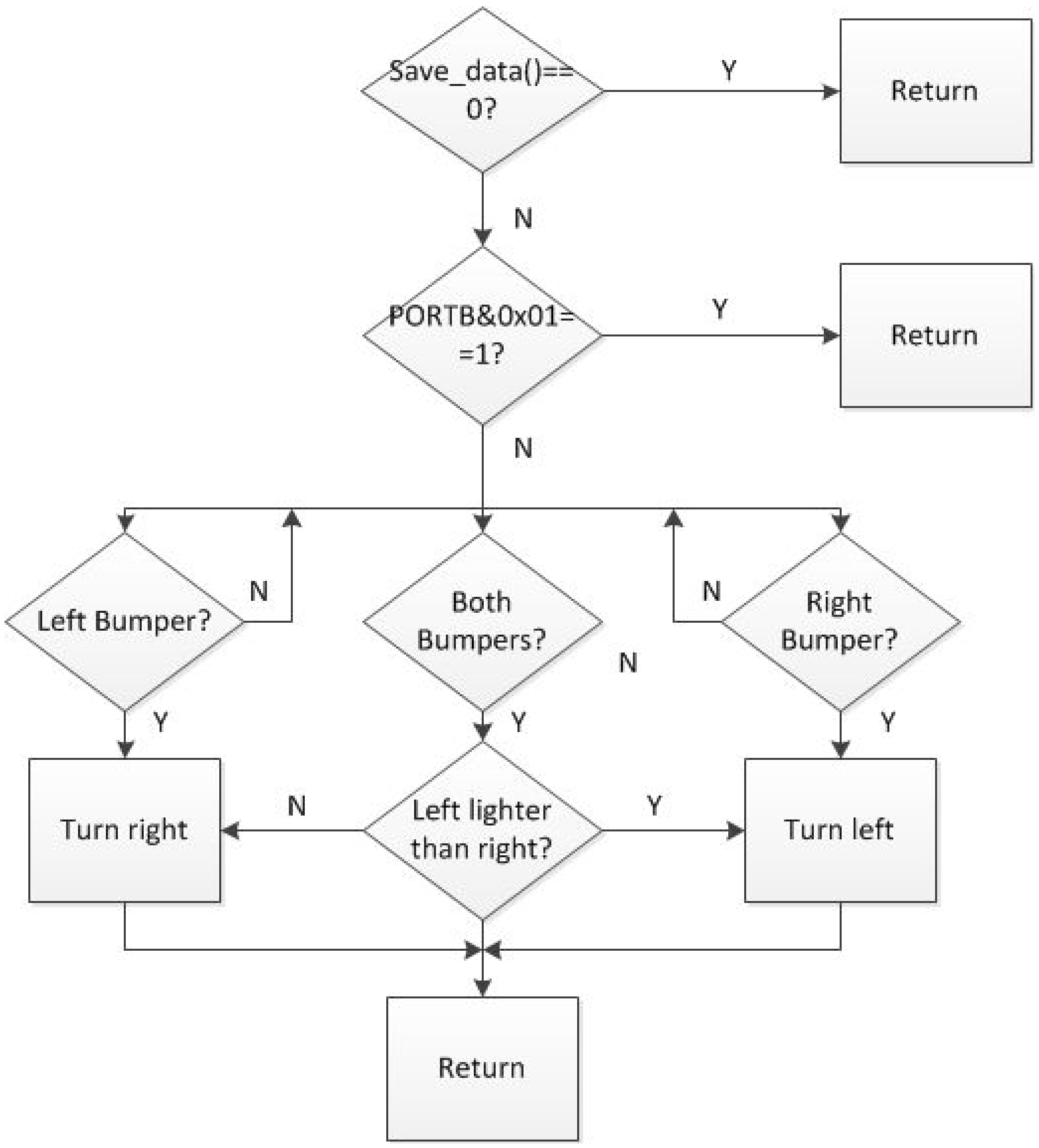
ii. Algorithm

```
void interrupt isr (void) // This function will be automatically called upon an interrupt
{
if (save_data ( ) ==0) // If interrupt was caused by real time clock, return. If
{return;} // interrupt was caused by bumper, save function variables
// being used by main() so they can be restored upon returning
// from interrupt.
pause (5); // Pause 5 msec to allow for bumper switch contact bounce
if ((PORTB & 0x01) == 1) // Confirm valid bumper interrupt
{
INTF = 0;
restore_data ( ); //Restore function variables used in main()
return;
}
output_low ('c', 0); //STOP the Motors! I’ve bumped into something.
output_low ('c', 5);
output_low ('c', 3);
output_low ('c', 4);
pause (50); //Always allow some time (50ms) for the motors to stop
/*YOUR CODE GOES HERE - I.e., Check which bumper(s) hit and take evasive motor
action. SEE IMPORTANT NOTES BELOW */
if ((read_input ('E', 0)) == 0&&read_input('E',1)!=0)
{
```

```
output_high ('E', 2);
   output_low ('A', 4);
   /*DO SOMETHING*/
   backward();
   pause(425);
   motorsoff();
   pivotleft();
   pause(125);
   motorsoff();
   forward();
}
else if ((read_input ('E', 1)) == 0&&read_input('E',0)!=0)
{
output_high ('A', 4);
   output_low ('E', 2);
   /*DO SOMETHING*/
   backward();
   pause(425);
   motorsoff();
   pivotright();
   pause(125);
   motorsoff();
   forward();
}
else if ((read_input ('E', 1)) == 0&&read_input('E',0)==0)
{
output_low ('A', 4);
   output_low ('E', 2);
   /*DO SOMETHING*/

   sensor3=read_adc(3);
   sensor5=read_adc(5);
   backward();
   pause(425);
   motorsoff();
   if(sensor3<sensor5)
   {
   pivotleft();
   pause(125);
```

```
    motorsoff();

    }
    else
    {
    pivotright();
    pause(125);
       motorsoff();
    }
    forward();
  }


 if ((PORTB & 0x01) == 1) // If all bumpers have cleared their obstacles
 {
 INTF = 0; // Clear interrupt flag bit
 output_high ('A', 4);
 output_high ('E', 2);
 }
 restore_data ( ); //Restore function variables used in main()
 return;
 }
```

iii. Flowchart


Save_data()==
0?
Y
Return
N
PORTB&0x01=
=1?
Y
Return
N
Left Bumper?
N
Y
Both
Bumpers?
N
Y
Right
Bumper?
N
Y
Turn right
N
Left lighter
than right?
Y
Turn left
Return