Introduction

The Robot

The robot project entailed the design, construction, configuration, and competition of a robot. This report details all of these steps as well as teaches first year engineering students the process of formulating design and discoveries into a written manner that other engineers, or anyone can read and understand. The project served as a proper introduction to basic engineering design skills such as teamwork, programming, using bench tools, soldering, software and hardware integration, and communicating the results.

The Design

The very first step in the project was filling out a Gantt chart detailing the responsibilities of each team member and when these tasks were to be accomplished. Anthony was responsible for the majority of the mechanical and electrical tasks in the Gantt chart. He performed almost all of the major hardware activities including robot chassis construction, mounting the motors, bumpers, lights sensors, etc., and constructed the Bumper Interrupt Support Board and the Floor Sensor Module. Greg was responsible for most of the software and integration. He performed nearly all of the major software activities such as writing the main loop code, developing motor and bumper control routines, and calibrating the software with the floor sensor and the target lights. Miranda was the Project Manager. She kept all of the documents and important files organized, and was responsible for creating most of the oral presentation as well assembling the report.

Project Requirements

The robot competition requires each group to design their own robot to battle an enemy robot in the arena. The goals of the competition are to detect the enemy's target lights and extinguish them. The robot must take into account the surroundings of the arena including the territory (home or enemy) as well as the obstacles. Most basically put, the robot must turn off the enemy's target lights as quickly as possible without extinguishing its own. Based on these basic requirements, the robot must contain mechanical, electrical, and software design that will accomplish all of these tasks. The major elements include the design and mounting of the bumpers, target light sensors, bumper interrupt board, and the floor sensor module as well as the programming that allows all of these elements to work together. The unique design of each robot comes from the strategic placements and configuration of each element. Each group must take into account the objectives of the competition as well as the rules and regulations and create the most effective robot design.

Materials and Tools

Material that must be returned and accounted for:

Components	Quantity
Chassis Subassembly - (Includes two gearboxes, two wheels, two piece PVC frame connected by screws and spacers, and a 9.6V Ni-Cad ***or*** NiMH battery (will be indicated on the chassis)	1
PIC Microprocessor Board - Record Unique S/N	1
Battery Charger (has battery plug on the end)	1
AC Wall adapter (Battery Backup)	1
Serial Cable (or USB A-B Cable)	1
USB to Serial Adapter (only if given Serial Cable)	1
Battery Connection Cable (re/black w/connector)	1
Interrupt Pin Connector with wire lead	1
Interrupt Circuit (Separate Envelope)	
Hex Buffer Integrated Circuit (IC) - 74LS07	1
14 pin IC Socket	1
4.7kΩ Resistor	1
Blank Printed Circuit Board	1
Bumper Switch Package (Separate Envelope)	
1/2" Long #2 Self Tapping Screws	4
Bumper Switches	2
PIC mounting Package (Separate Envelope)	
1/4" Nylon Spacers	4
1/2" Long #4 Self Tapping Screws	4

Expendable materials (but return if not used and still in original condition)

5 1/2" x 5 1/2" x 3/16" Gray PVC	1
5 1/2" x 5 1/2" x 1/8" Gray PVC	1
5 1/2" x 5 1/2" x 1/16" Gray PVC	1

FSM Package (Separate Envelope)

High Current White LED	1
220 Ohm Resistor	1
20k Ohm Resistor	1
2N2222 Transistor	1
Blank Printed Circuit Board	1

Light Sensor Package (Separate Envelope)

Cadmium Sulfide Light Sensors	3

General supplies will be available in the Lab

Self-tapping screws for connecting robot body parts
Machine screws and machine nuts for connecting robot body parts Washers for use in conjunction with screws
Electrical tape (For electrical connections only)

Tools:

Typical tools required to complete your robot project will be provided. In addition to the lab bench tools (drill press, band/scroll saws, belt sander, etc.), a tool box with hand tools (screwdrivers, wire cutters, soldering iron/solder, etc.) will be provided. An inventory of the tools will be posted on the box. To get a tool box, someone in the group must surrender a student ID to the section T/A. Upon returning the tool box and after a successful inventory, the ID will be returned.

System Design

The primary goals of the project are to design the most effective robot possible and win the competition. Both the hardware and software components were designed strategically to carry out these objectives. Before construction began, multiple design features were considered and conceptually thought out in relation to the objectives of the competition. This Pugh matrix took into account the major hardware elements such as the number and shape of the bumpers, the number of target light sensors, the location of the floor sensor module, and the height of the beacon light sensor.

Number of Bumpers		3 Bumpers		2 Bumpers		3 Bumpers		4 Bumpers	
Shape of Bumpers		Trapezoidal - Hinged		Triangular - Hinged		Rounded front		3 ft trapez; 1 bk round	
Number of Target Light Sensors		2: 1Lt, 1Rt		2: 1Lt, 1Rt		2: 1Lt, 1Rt		2: 1Lt, 1Rt	
Height of Beacon Sensor		8 in. above roof in back		8 in. above roof in back		8 in. above roof in back		8 in. above roof in back	
Floor Sensor Module Photo Resister Location		Rear		Rear		Front		Rear	
Any other paramter you think is important		Layered, speed		Layered, speed		Layered, speed		Layered, speed	
Acceptance Criteria	**Weight (by %) of Acceptance Criteria (apply last)**	**Importance of Conceptual Design in meeting Acceptance Criteria**	**Weighted Score**	**Importance of Conceptual Design in meeting Acceptance Criteria**	**Weighted Score**	**Importance of Conceptual Design in meeting Acceptance Criteria**	**Weighted Score**	**Importance of Conceptual Design in meeting Acceptance Criteria**	**Weighted Score**
If Robot turns back to home territory, program tells it to turn back around	5%	10	0.5	10	0.5	10	0.5	10	0.5
Bumpers sense when touch a wall	20%	8	1.6	5	1	8	1.6	10	2
Doesn't confuse the navigation light with target lights	30%	8	2.4	10	3	8	2.4	8	2.4
Workmanship - Structurally Sound	5%	8	0.4	10	0.5	8	0.4	5	0.25
Finds way to Enemy Territory	5%	10	0.5	10	0.5	5	0.25	10	0.5
Extinguishes both target lights in 35 seconds	5%	8	0.4	10	0.5	8	0.4	5	0.25
After crossing over, the sub system detects light 1 and extinguishes it. Then it detects light 2 and extinguishes it.	15%	10	1.5	10	1.5	10	1.5	10	1.5
Doesn't cross back to friendly territory after crossing into enemy territory	15%	10	1.5	10	1.5	5	0.75	10	1.5
Total Percentage =	100%		8.3		8.5		7.3		8.4

Design 1

The main design element that sets this design apart from the others is the fact that there are 3 bumpers and they are trapezoidally hinged. The rest of the design elements were already decided upon in advance by the group. This design was in very close contest with two other designs, since it was so similar. However, ultimately it was decided upon that the extra bumper would slow down the robot, therefore slowing down the robot's ability to extinguish the opponent's lights.

Design 3

This design's major feature was changing the location of the floor resistor module from the rear of the robot to the front. This would heavily impact the robot's ability to find enemy territory and then extinguish the target lights.

Design 4

The final design nearly beat out the winning design, the only difference being 4 bumpers instead of 2. It was determined that designed a 4 bumper robot would not only slow down the

construction process, but also the actual speed of the robot which then impacts the time it takes to extinguish the enemy target lights.

The Winning Design

The complexity of the bumpers in this specific robot allows it to sense a direct hit from both sides as well as the front. The bumper system is composed of 2 bumper switches with the bumper designed in a trapezoidal shape creating 3 possible "hit" scenarios: the left, the right, and the middle—which triggers both bumper switches. After some testing of the robot with the software, a guide was added to the bumper system to keep the bumpers centered and allow for more stabilization. The bumper design was the most heavily focused on, the rest of the design was already decided on.

Countermeasures Employed

This group chose not to use any countermeasures. It was determined that countermeasures would have been ineffective; the effort and power put into the countermeasure may have taken away from the actual important part of the robot. The risk was not worth the potential benefit.

I/O Pins to Robot Functions

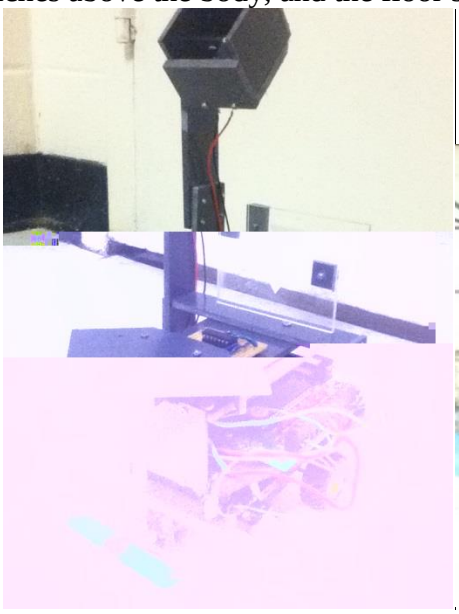
	Port	Pin/Channel Number
Light Sensors		
Floor	A	0
Target	A	1
Navigation	A	2
Bumpers		
Right	E	0
Left	E	1
Floor Sensor Circuit		
Floor LED Switch	D	7

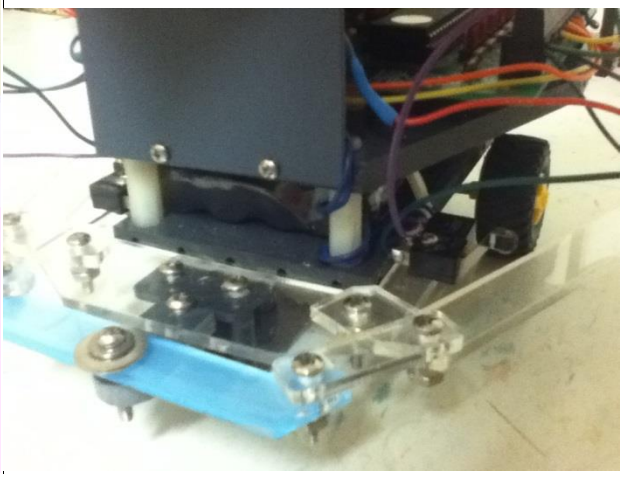
The PIC Microcontroller board has 33 input/output pins to interface with other circuits and sensors attached to PIC board. The motor and LED ports/channels were determined by Stevens and not adjustable by the group. Therefore the group was free to place the bumpers in the table above as it best saw fit.

Mechanical Design

Design

Before any major programming integration and strategizing could begin, the mechanical design and construction needed to be completed. After filling out the alternative design matrix, the design of the robot was decided upon. It was determined that the most effective robot would have two bumper switches, surrounded by a trapezoidal, hinged bumper system, two target light sensors (one on each side), the beacon light sensor set in the back of the robot approximately eight inches above the body, and the floor sensor module in the back of the robot.

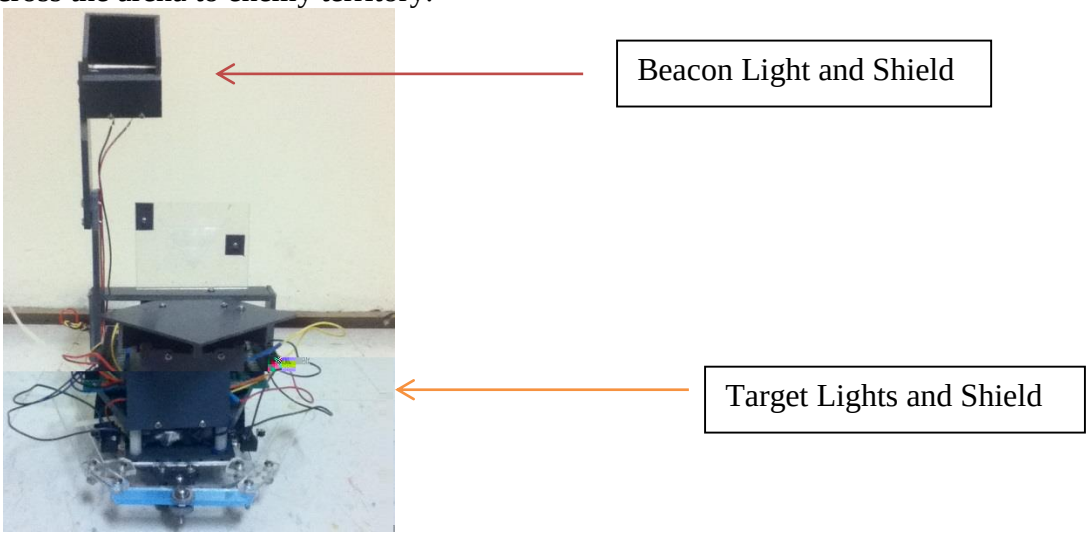




Mounting

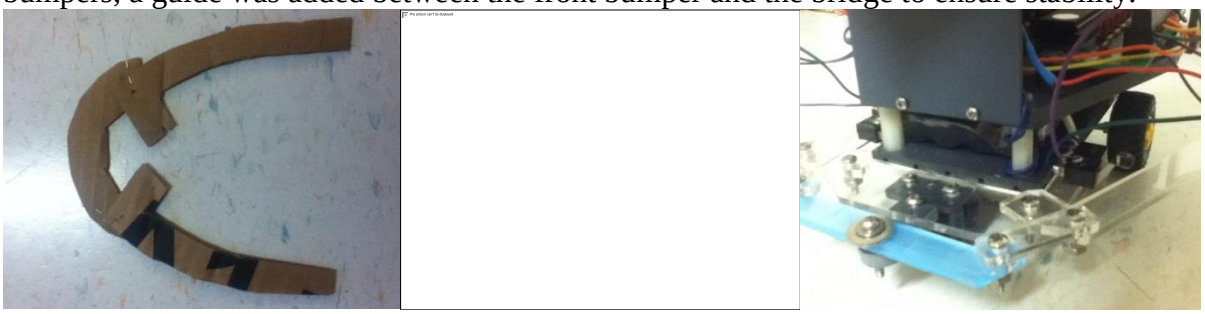
Mounting the light sensors required a lot of extra construction using the provided gray PVC and bench tools. The target lights sensors were mounted on the front with light shields constructed out of the PVC and screwed into the body of the robot. These shields help to capture the maximum amount of light so the sensors can detect the location of the target lights, and extinguish them. The beacon lights sensor is mounted in the back right corner of the body of the robot. Because the gray PVC was not long enough to reach the desired height, the tower has two separate pieces of PVC put together with four screws. At the top of the tower is a small, open, box (also constructed out of the PVC) set at an angle of approximately 45 degrees. This is used to shield any extra light from reaching the sensors. The beacon light sensor detects where the

beacon light, the large light bulb attached to the ceiling of the arena, and initially helps guide the robot across the arena to enemy territory.



The floor sensor module was attached to the robot on the bottom, back side. The floor sensor flashes an LED light and takes a reading of the floor of the arena. Based on these numerical readings, the robot is able to tell whether it is on the correct side of the arena. It was simply screwed onto the bottom of the root.

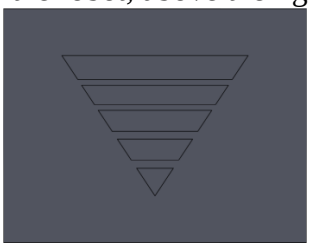
The bumpers were mounted in the front and wrapped around the sides to provide the most coverage to determine a "hit." The first bumper design had a rounded front, and two identical pieces hinged to the front and wrapped around the sides of the robot. However, they would have been ineffective as the side pieces would have been unable to register a hit due to the L-shape. The next design was constructed out of paper, then designed and dimensioned in Solidworks. These accounted for the ineffective L-shape in the previous design, by bridging the two identical side pieces together. Once the drawings were created in Solidworks, they were cut out using a laser cutter to ensure precision and workmanship. Upon testing the newly cut bumpers, a guide was added between the front bumper and the bridge to ensure stability.



Design 1 Design 2 Final Result

Logo

The robot's logo was designed based on the name of the robot, Dorothy. Drawn in Solidworks, the logo is a simple geometric representation of a tornado. It was mounted on the front of the robot, above the light sensor shields.



Software Design and Coding

Process

The software design was extremely critical to the success of the Robot. The first thing that was considered when designing the robot was focusing on what the robot was supposed to do. Although this seems like trivial question, it was necessary for the group to lay out what was necessary for the robot to accomplish and what importance was assigned to each respective task. For example, since the point of the competition is to put out the target lights, it is imperative to make sure that the robot does not confuse the beacon light with the target light as the result could lead to the robot focusing on the wrong light throughout the competition. The ability of the sensor to distinguish between the beacon light and the target light was assigned at 30% importance on the Gantt chart (see page 43). Another area of importance that given a significant amount of focus was the bumper design and the ability for the bumper to recognize whether or not it had been hit. The importance of the bumper effectiveness was ranked at 20%. Two more areas that were of importance were the bottom floor sensor module to make sure that the robot does not cross back over from the enemy side and the robot's ability to extinguish both of the enemy lights. Both of these tasks were placed at 15% importance. Finally, three tasks were given 5% importance; indicating that some, but relatively little importance was placed on these topics. These three tasks include: workmanship/pride in work taken when creating the robot, the ability of the robot to find its way into enemy territory, and speed of extinguishing the target lights. These factors were the most important influences when creating the code.

This project required the use of the C programming language, which is a very structured language so it was important that the code reflected the language in that it was very structured and clear. Before writing any worthwhile code, a flowchart was created that described the process that would be followed when writing the code (see page 47). First, several subroutines were created in the code using "void functions." Once the basis of the code was down, the specifics of the code were tested.

Testing

The testing process is extremely important because it can cause variations that the design and coding process did not predict. For example, when the robot was told to go straight, it curved to the left because of an imbalance in the motor strengths, even though the motors were set to equal values in the code. Another important test that had to be completed was figuring out how long of a pause statement would be needed to propel the robot one foot. This could only be done with a trial and error process. To figure this out a ruler was placed on the area the robot would be competing on (so that the coefficient of friction would be the same) and experimented with values ranging from 200 to 1000 until we figured out which value would propel the robot one foot. The same experiment was used for figuring out how long we had to "pause" the robot when pivoting or turning. See table of values on page 41.

Program Requirements

The programming required an IDE, compiler, and a bootloader to design the software. The IDE was used to write and edit the C programs (.c file) using an editor that has syntax coloring aids. The Hi-Tech Compiler was used for compiling the C program into assembly language (.as file) and getting information on errors that may exist in the program. Every time a program was created or changed a ".hex" file needed to be downloaded. The ".hex" file was created by the complier/assembler into the robot's PIC microcontroller. Using the "bootloader" one is able to accomplish this without having to leave the IDE and locate the bootloader program manually each time one wants to download a revised robot program. One is able to tell the IDE where the .hex file was located so that we were able to use the bootloader, simply, by pressing a button in the IDE.

The following were required for the software design of the robot:

- Subroutines
- An interrupt function
- A main function
- The real time clock

Subroutines

Subroutines were important because they allow the program to access code over and over without have to constantly rewrite it. Tasks the program used often were created into subroutines. It's important to note that no subroutines were used in the "bumperInterrupt" function because this would cause a stack overflow. A stack overflow happens when the queue of code exceeded a certain number which makes the microprocessor unable to manage all of the code. In this case, the "stack" limit is 6 functions. Here, there are 12 subroutines.

1. void pivotLeft();

Void pivotLeft allows the robot to pivot left. A pivot is executed when both wheels of the robot are spinning in opposite directions. It accomplished this by giving output 0, 0 volts of

electricity with an “output_low” statement while setting the voltage on its corresponding output (output 5) to 5 volts with an “output_high” statement. By setting the opposite statements on motor two (output_low on output 3 and output_high on output 4), the motor is forced into reverse which satisfies the condition of a pivot.

2. void pivotRight();

Void pivotRight allows the robot to pivot right. A pivot is executed when both wheels of the robot are spinning in opposite directions. It accomplished this by giving output 0, 5 volts of electricity with an “output_high” statement while setting the voltage on its corresponding output (output 5) to 0 volts with an “output_low” statement. By setting the opposite statements on motor two (“output_high” on output 3 and “output_low” on output 4), the motor is forced into reverse which satisfies the condition of a pivot.

3. void motorOff();

Void motorOff allows us to turn the motor off. A motor off command is executed when all the motors are not spinning. It accomplished this by giving all four output’s “output_low” statements which stops all voltage from going to the motor.

4. void redLedOn();

Void redLedOn tells the robot to turn the red led on. This is useful when trying to register a bumper hit when testing. A redLedOn is accomplished by giving output two an “output_high” statement which allows voltage from to go to the motor.

5.void redLedOff();

Void redLedOff tells the robot to turn the red led off. This is also useful when trying to register a bumper hit when testing. A redLedOff is accomplished by giving output two an “output_low” statement which allows voltage from to go to the motor.

6. void forward();

Void forward tells the motor to go forward. A void forward function used in conjunction with a pause (x amount) statement propels the robot forward a certain distance. A forward command is executed when both motors are spinning in the same direction which causes the wheels to spin in the same direction (clockwise). It accomplished this by giving outputs 5 and 3 “output_high” statements which send 5 volts to the ports. Outputs 0 and 4 are given “output_low” statements which stops voltage from going to the outputs.

7. void back();

Void back tells the motor to go back. A void back function used in conjunction with a pause (x amount) statement propels the robot back a certain distance. A back command is

executed when both motors are spinning in the same direction which causes the wheels to spin in the same direction (counterclockwise). It accomplished this by giving outputs 0 and 4 “output_high” statements which send 5 volts to the ports. Outputs 3 and 5 are given “output_low” statements which stops voltage from going to the outputs.

8. void turnLeft();

Void turnLeft allows the robot to turn left. A left turn is executed when the right wheel is moving clockwise while the left wheel is not. It accomplished this by giving output 0, 0 volts of electricity with an “output_low” statement while setting the voltage on its corresponding output (output 5) to 5 volts with an “output_high” statement. By giving motor two (output 3 and 4) “output_low” statements, it stops the left wheel from turning, which defines a left turn.

9. void turnRight();

Void turnRight allows the robot to turn right. A right turn is executed when the left wheel is moving clockwise while the left wheel is not. It accomplished this by giving output 3, 0 volts of electricity with an “output_low” statement while setting the voltage on its corresponding output (output 4) to 5 volts with an “output_high” statement. By giving motor 1 (output 0 and 5) “output_low” statements, we can stop the right wheel from turning, which defines a left turn.

10. void yellowLedOn();

Void yellowLedOn turns the yellow led on. This is useful when trying to register a bumper hit when testing. A yellowLedOn is accomplished by giving output four an “output_high” statement which allows voltage to go to the motor.

11. void yellowLedOff();

Void yellowLedOff turns the yellow led off. This is useful when trying to register a bumper hit when testing. A yellowLedOff is accomplished by giving output two an “output_low” statement which allows voltage to go to the motor.

12. void bumperInterrupt();

The bumperInterrupt function is one of the most important and unique functions because it defines the strategy. The reason for the bumperInterrupt function is to “interrupt” the main function if it tripped by a change in voltage in the appropriate output. For example, it is important to have an interrupt function connected with the bumpers since the interrupt function will stop the main function if it detect a bumper hit. The interrupt function will therefore be useful in stopping the robot from perpetually crashing into a wall. The interrupt function works by first reading the signal from the output. If it reads 0 then it will assume that there was no bumper hit and continue on with the main function. If it reads 1 then it will confirm the bumper hit and proceed with the written function. If the bumper hit is confirmed, the pic will first stop

the motor. The interrupt will then decipher which bumper switch was flipped using an "if…else" statement. If the output assigned to the left bumper reads a 1 and the output assigned to the right bumper reads a 1 then both bumpers were hit and the robot will pivot 180 degrees and return to the main function. If the output assigned to the left bumper reads a 1 and the output assigned to the right bumper reads a 0 then the left bumper was hit and the robot will back up for 650 milliseconds (or 1 foot), pivot 125 milliseconds (or 45 degrees) to the right and return to the main function. If the output assigned to the left bumper reads a 0 and the output assigned to the right bumper reads a 1 then the right bumper was hit and the robot will back up for 650 milliseconds (or 1 foot), pivot 125 milliseconds (or 45 degrees) to the right and return to the main function. Finally, the before returning to the main function, the interrupt will clear the cashed bit to ensure that the microprocessor does not continually read the bumper hit as a confirmed hit.

Pre-defined Subroutine Functions:

1.) void configurePIC ();
2.) void motorspeed (motornum, speed);
3.) void output_high (port, bit);
4.) void output_low (port, bit);
5.) unsigned int read_adc (channel);
6.) char read_input (port, bit);
7.) void putdata (data);
8.) void putchar (data);
9.) void pause (data);

1.) Name: configurePIC ( );

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. Additionally the serial communications port (RS-232 or USB) to the computer (laptop) is setup and configured for 19.2K Baud. This function MUST be called once in the main function before entering the continuous while (1) loop.

2.) Name: motorspeed (motornum, 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 the robot. This function limits the low speed to 30% to prevent a motor stall condition that would cause excessive current drain and possible damage to the hardware. Calling this function with a speed of less than 30% will be ignored. This function does not control motor rotation direction, or motor on/off status. This function must be called at least

once for each motor to set an initial speed; otherwise the motor will not turn. Once set, the motors will remember the last speed setting and use that setting whenever they are turned on, in either direction.

This robot uses the motorspeed command to alter the values of the motor. When both motors were running at the same value(90) the robot would drift towards the left so an adjustment was necessary to ensure that the robot would travel in a straight line. The final values of the robot in order to make it go straight were 85 for the left motor and 80 for the right motor.
Inputs out-of-range will result in a return with no action. No error condition is reported back.
Output Parameters: None

3.) Name: 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) bit = 4 for port A; 0-7 for ports B and D; 0, 3, 4, or 5 for port C, 2 for port E.

Inputs out-of-range will result in a return with no action. No error condition is reported back.

Output Parameters: None.

Output_high functions were used in conjunction with output_low functions in nearly all of the code. Output_high and output_low told the robot what direction to move in. For example, a forward function and a backwards function were created as well as a motor off function which served as the basis for the main strategy. The output functions also allowed us to alter the direction the motor was spinning in, which allowed the robot to perform more advanced functions such as turns and pivots.

4.) Name: 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.
Input Parameters: port = 'A', 'B', 'C', 'D', 'E' (lower case is also accepted) bit = 4 for port A; 0-7 for ports B and D; 0, 3, 4, or 5 for port C, 2 for port E
Inputs out-of-range will result in a return with no action. No error condition is reported back.
Output Parameters: None

Output_high functions were used in conjunction with output_low functions in nearly all of the code. Output_high and output_low told the robot what direction to move in. For example, a forward function and a backwords function were created as well as a motor off function which served as the basis for our strategy. The output functions also allowed us to alter the direction the motor was spinning in, which allowed the robot to perform more advanced functions such as turns and pivots.

5.) Name: 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 is used to read the output of the light sensors.
Input Parameters: channel = 0 – 4. Physically all five channels are on port A. “configurePIC ( )” takes care of setting up these analog input channels. Channel inputs that are out-of-range result in a return value of “65535” (max value - hexadecimal 0xFFFF).
Output Parameters: unsigned int = A sixteen bit unsigned integer representing the voltage present at the specified analog channel. Note you must define this variable as an “unsigned integer”. The number will typically be in the range of 00000 – 50000, representing 0- 5 volts. Upon return to the calling program, the value will be stored in the variable specified by your program. If an invalid input channel is entered, the number 65535 (hexadecimal 0xFFFF) is returned. The unassigned int function was used with all of the robot’s sensors as it basically updated the values of the respective values constantly as they were being updated.

6.) Name: 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; 1 indicates the pin is connected to 5 VDC.
Input Parameters: port = ‘D’, ‘E’ (lower case is also accepted) bit = 0-7 for ports D; 0, 1 for port E
Inputs out-of-range will result in the value of “255” (Hexadecimal 0xFF) being returned.
Output Parameters: char = An eight bit unsigned integer equal to 0 or 1 (the logic level of the requested input port/bit) or 255 if the input port/bit was out-of-range.

7.) Name: putdata (data);
Purpose: Sends the ASCII equivalent of a sixteen bit integer (max) over the RS-232 serial port or USB port. It is used to send data to a terminal program so it can be viewed. This is useful for viewing the output of your light sensors, or for debugging purposes by viewing the value of any variable at strategic places in your program. This function assumes that “configurePIC ( )” has previously been called to setup the PIC RS-232 port for operation/baud. It also assumes the COM port on the receiving computer has been setup for 19.2K Baud, 8 data bits, 1 stop bit, no parity, and no flow control.
Input Parameters: data = any 8 or 16 bit variable or constant in your program
Output Parameters: None returned to the calling program, however characters will be sent over the RS-232 or USB serial port.
The putdata was used when calibrating the light sensors for the robot. The put data function displayed the values that the light sensor was reading which was crucial during integration testing.

8.) Name: 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. This is useful for commenting or delineating data sent by "putdata". This function does not support character strings. This function assumes that "configurePIC ( )" has previously been called to setup the PIC RS-232 port for correct operation and baud. It also assumes the COM port on the receiving computer has been setup for 19.2K Baud, 8 data bits, 1 stop bit, no parity, and no flow control.
Input Parameters: data = a single ASCII character, or decimal 13 (carriage return)
Output Parameters: None returned to the calling program, however the specified character will be sent over the RS-232 or USB serial port.
Putchar was used to help organize all of the values that were received from putdata. Putchar associated each sensor with a letter which would then appear on the terminal when reading the data.

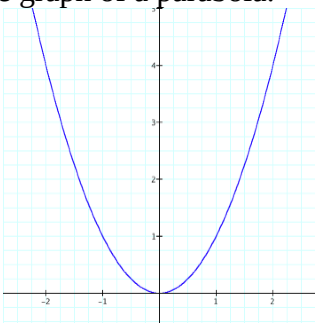
9.) Name: pause (data);
Purpose: To introduce a delay in the program on the order of milliseconds, 1 millisecond minimum, 65,535 milliseconds (65.535 seconds) maximum. This function operates by timing an execution loop. This function will not return to the calling function until the specified time interval has been reached. Care should be exercised when using program delays as all other software (except the bumper interrupt routines) will not execute until the end of the delay is reached. Small pauses are used primarily to control motor movements. Pauses greater than 1000 milliseconds (1 sec) should be avoided!
Input Parameters: data = a 16 bit unsigned integer variable or constant in the range 1-65535
Output Parameters: None

The pause function was used to specify how long a specific function was to run. For example, it was used to determine how long the robot would need to "Pause" for in order to move forward a specific distance. See table of values on page 41.

Strategy

The use of the mathematics concept, the minimum, is integral (pun intended) to the software design as well as to the strategy. The theory that the robot will take an initial reading (called "previousValue") and then pivot left a small amount and take another reading (called "currentValue") and it will pivot left one more time (the same small amount) and take one final reading(called "finalValue"). Comparing these readings, the robot will determine the lowest value of the three. It is looking for the lowest value because, the lower the value; the brighter the light. For example, if the value of "previousValue" is lower than "currentValue" and "finalValue" then it will be able to determine that the brightest light (the beacon light) is on the right. In the case the beacon light is on the right, the robot will next to make a full circle in order

to determine where the beacon light is. Fortunately, one revolution of the robot, using the minimum concept, will take approximately 2 seconds so even if the robot starts in a less than ideal position (where it is just to the right of the target light), it will be able to rebound quickly. The idea is to get the robot in a position where, the value of "previousValue" and that of "finalValue" are greater than the value of "current value". If this condition is met, the robot will have found it minimum (or the position where the light is at it highest). This can be illustrated with the graph of a parabola.



Notice how on the graph of the parabola, that from the left, the parabola decreases to 0 and from the right it approaches 0, this illustrates the concept of a minimum (where in this case f(0) is the minimum). The minimum will not be 0 but it will be the lowest value that the light sensor can read while the robot has not moved vertically or horizontally from its position. The robot frequently reuses the minimum concept not only while searching for the beacon light, but also while searching for the target lights.

The use of counters is more of a failsafe of the robot. Initially, the counters were implemented in order to stop the robot from being trapped in the corner of the arena. Combined with the real time clock, the counters could be used to determine how many bumper hits the robot as incurred throughout a period of time which could help the robot from becoming trapped in the corner of the arena. Thanks to the minimum strategy it is unlikely that the robot will become trapped in the corner of the area. However the counter strategy is still implemented as a failsafe for the main, minimum strategy.

Much of how the program used interrupts was discussed in the void bumperInterrupt of the function portion of the software section. What was not discussed is how the interrupts actually worked. An interrupt works at the hardware level which means it bi-passes any of the code that was written. For example, the computer interrupt was used in conjunction with the bumpers because bumper collisions are sporadic and need to be dealt with immediately before the robot can proceed with its intended function. For this, an interrupt is necessary as it allows the robot to never have to "check" for a bumper hit.

Building and Connecting the Bumper Interrupt Support (BIS) Board

I. Understanding Operation of the Bumpers

At minimum, it is recommended that the design of your "Collision Avoidance Subsystem"include two bumpers, one located front left and the other front right. The physical design and attachment of the bumpers is up to you. You will be provided with two electrical bumper switches that will attach to the robot using two self tapping screws. The screws will go into holes that you locate and drill into a mounting platform that you design. An assortment of expendable plastic (see "Resources Provided" document) that can be cut and attached to the robot chassis will be provided. Keep in mind you cannot drill into, or glue onto, the two ¼ inch thick plastic robot chassis pieces, but you may attach other plastic pieces to them using the predrilled edge holes and screws. The actual bumper design (the part that will contact an obstacle and close the electrical switch) is also up to you. Should your design include more than two bumpers, tell your T/A so that additional parts can be provided.

The bumpers operate by closing the contacts of an electrical switch. The switches have three connection terminals. When wiring the switches use the two terminals marked common, C, and N.O. (normally open). The third terminal is not used in our application. The PIC microcontroller allows for easy connection of two bumpers. See "Connecting to the PIC Board". These bumper connections are assigned to Port E bits 0 and 1. Interrogating these bits using the "read_input" function will tell you if a particular bumper has been activated. If "read_input" returns a value of "zero", the bumper is activated. If it returns a value of "one" the bumper is clear of obstacles.

One way to check for collisions is to periodically interrogate the status of the bumper inputs to see if any of them has activated. This would be done periodically at strategic places within the main program loop. There are disadvantages to this technique, particularly:

1. You may not "strategically" look for bumper closures within your main loop (i.e., you will not be checking for closures at times when you should)
2. When you use program delays (i.e., using the function "pause") the microcontroller will be busy counting down the delay and will not execute other statements in the program until the delay is complete. If the motors are turning when you execute a delay (which they typically are) your robot can be crashing into something and even though the bumper(s) activate, your program will not be checking them because the microcontroller is tied up counting down the delay. This results in excess stress on the motors/gears/bumpers and a larger current drain through the electronic circuits. This can significantly reduce your battery charge, and may also trigger protection circuitry on the PIC microcontroller which will cause your robot to slow down or stop working.

A better way to detect collisions is to use the "computer interrupt" capabilities of the PIC microcontroller.

II. What is a Computer Interrupt?

A computer interrupt is a hardware driven event. That is, when the value (voltage) on a particular input line of the microcontroller changes state (for example when a bumper switch closes) the internal hardware of the chip will *immediately* recognize the event, *immediately* stop whatever it was doing (even counting down a delay), save the current processing environment (so it can return there later) and jump to a certain location in memory where the "interrupt service routine" is located. The interrupt service routine is a function named "**void interrupt isr (void)**", and it will have your particular bumper handling code that will check to see which bumper(s) caused the interrupt and then take appropriate motor control actions to get around the obstacle. Once the bumpers indicate that the obstacle has been cleared, the interrupt function will return to the main program, restore the environment that was saved when the interrupt occurred, and continue exactly where it left off. The result is that as soon as any bumper activates evasive action as determined by the design of your bumper control routines will occur immediately. The crashing/motors grinding scenario will be avoided.

III. Implementing Bumper Interrupts on the Robot

Implementing interrupt driven bumpers on the robot will require both special hardware and software considerations.

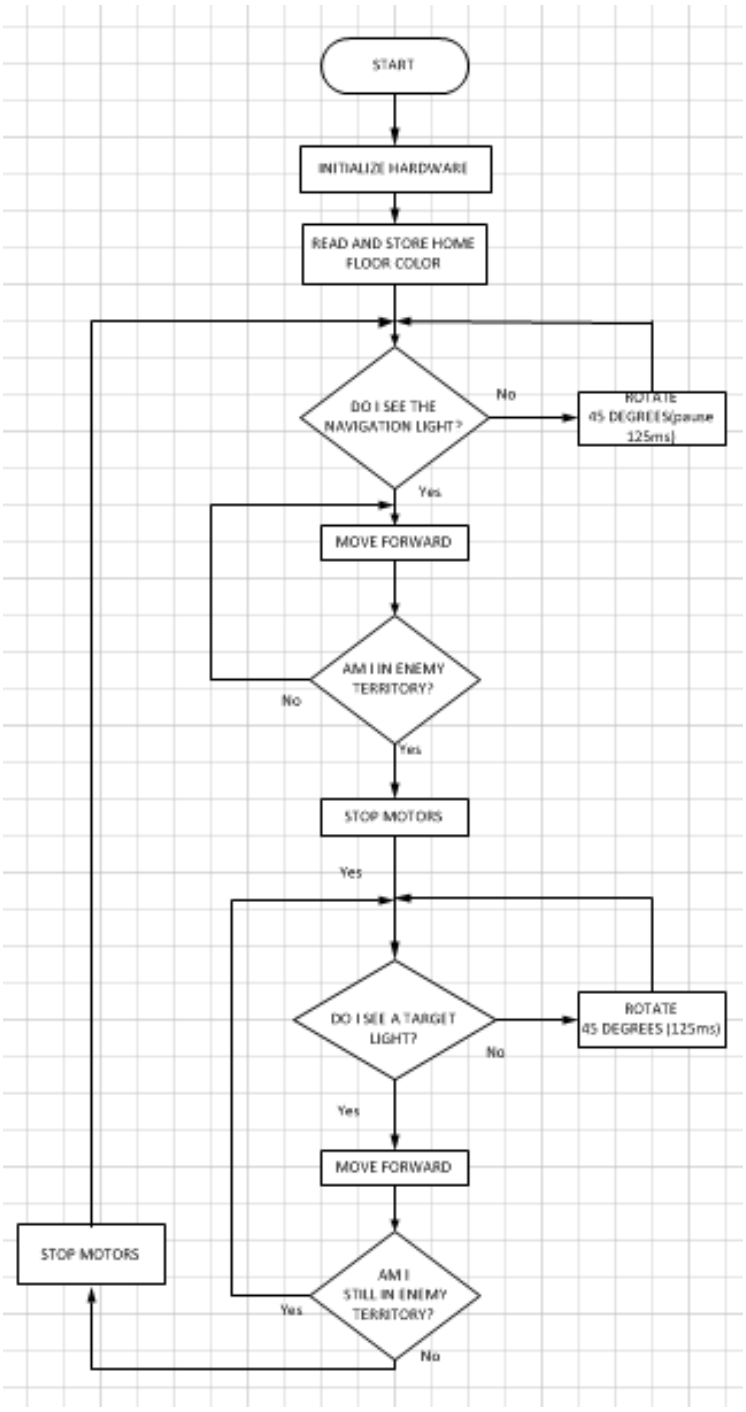
Hardware Required

The PIC microcontroller has one connection called the "External Interrupt" input. This input is on Port B, Bit 0. Port B is located on the blue plug-in connectors next to the Port D screw-in terminal strip. To access it, a wire with a plug-in pin will be provided. The way the PIC is configured, a signal (voltage level) transitioning from high (5 volts) to low (ground) will cause the interrupt to occur. Since the PIC has one external interrupt input, but your robot has at minimum two bumpers (which appear on Port E Bits 0 and 1), and since we must be able to determine which bumper was activated (left or right) we cannot simply connect the bumper signals together and then route them to the interrupt line. To resolve this problem we will use a simple electronic chip called a buffer. This chip will be located in a socket that you will solder onto a small piece of printed circuit board that you will locate somewhere on your robot and then connect to the bumper and interrupt inputs. **This printed circuit board is called the "Bumper Interrupt Support" (BIS) board.**

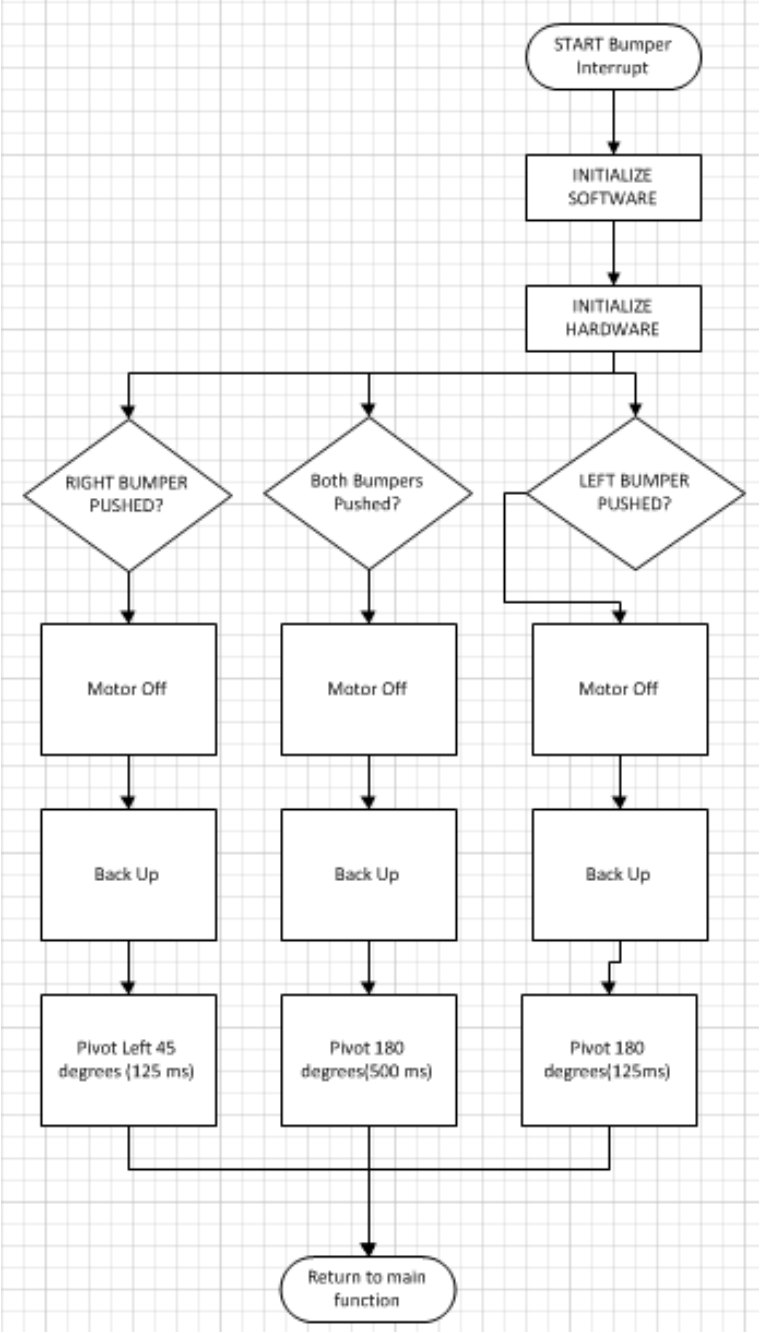
Physically, the buffer comes packaged as a 14 pin dual in-line chip. Internally the chip actually contains six separate buffer circuits. If you have two bumpers, you will use two of those buffer circuits connected accordingly, plus the chip must get power (VCC) and ground (GND) to operate. VCC and GND are available on the PIC board black screw-in connection strip that also has Port D on it. Additionally a pull-up resistor needs to be located on the BIS. One of the

resistor leads will connect to the interrupt line, and the other to VCC. So, assuming you have two bumpers, the BIS board will have five wires coming off of it that get connected to the PIC Board.

The real time clock was used in order to ensure that our robot does not get trapped in a corner. While, somewhat unnecessary due to the minimum strategy, the real time clock acts as a failsafe in case the robot somehow gets caught in a corner.



Main Flowchart



Bumper Flowchart

Flowcharts

The flowchart is divided up into two functions: the main function and the bumper interrupt function.

The main function is a perpetual loop that is always running as it causes the robot to run. The first thing we must show on the flowchart is that we initialize the hardware. This is a crucial part of the robot design as it sets the default value for the motor speed as well as making sure that the floor sensor light stays on. The next thing the flowchart tells us is that the robot will store the value that the floor sensor records which will tell us which side the robot starts on, which is useful for making sure that it does not cross back over from enemy territory. After the light value is recorded, the while loop can begin. The first question the program will "ask" is "Can I see the target light?" The reason for this is so the robot can get its bearing and attempt to determine the starting position. From there, the robot will perform one of two actions. If the robot detects the beacon light, it will proceeded forward, if the robot does not read the beacon light then it will pivot in 45 degrees, or 125 milliseconds, increments until it detects the light. In either case, the robot will end up moving forward in the direction of the beacon light. Once the robot detects that there has been a change in the floor sensor value, the robot will determine that it has crossed over into enemy territory. From there the robot will stop the motor and turn in 45 degree increments until it detects one of the two target lights. Throughout the flowchart, the main function will check to ensure that the robot stays in enemy territory. If the robot crosses back over, the main function will tell the robot to stop and turn around so that it can renavigate its way to enemy territory.

The bumper interrupt is unique in that it activates whenever the robot bumps into an object. The details of bumper interrupt, such as specifically how it interrupts the main function, are described in the "void bumper interrupt" section. The "bumper interrupt" starts by initializing both the hardware and software. It is crucial that the software is initialized because in order for the pic to recognize that the interrupt function has been enabled we must declare it during the software initialization period in the code. Once initialized, the bumper interrupt will not be activated until the bumper switch is tripped. The bumper interrupt will start by "asking" which bumper(s) have been tripped. If both bumpers have been tripped then the robot will stop the motor, back up and then turn around 180 degrees, or 500 milliseconds. If the left bumper is tripped, the robot will stop the motor, back up and pivot 45 degrees to the right. Likewise, if the right bumper is tripped, the robot will stop the motor, back up, and pivot 45 degrees to the left.

Downloading the Software

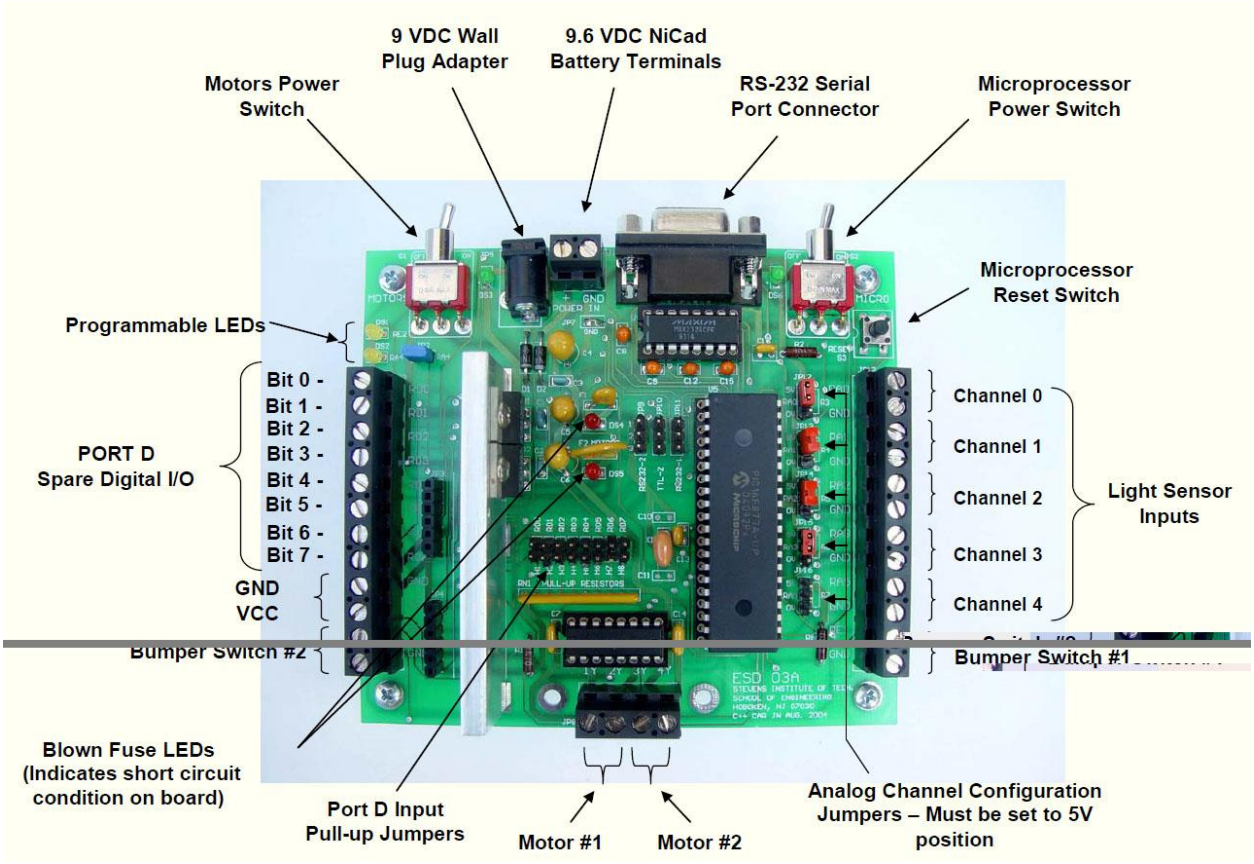
It was decided that Greg would be this group's programmer and one of the first steps of the project was to download the software to make programming the robot a possibility. The software was provided on Moodle and then the group followed the instructions to download it

onto Greg's computer. Once the software was completely uploaded the group began to learn about the software and the use of PIC boards.

Electrical and Wiring Design

The PIC board

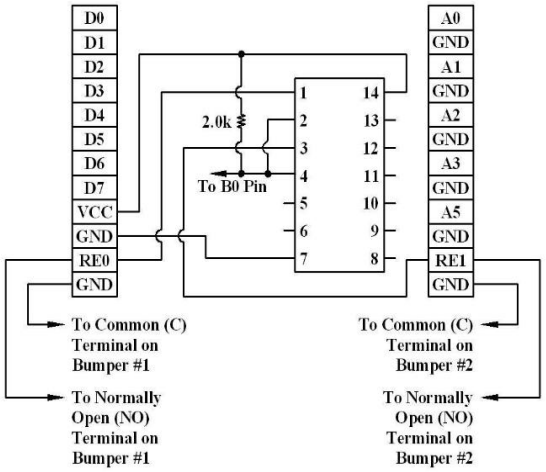
The pic board is the central computer controller for the robot, connections are made to the pic-board via wires and the input/output terminals are on the board. The PIC microcontroller circuit board was designed and built by Stevens Institute of Technology. It was designed around the PIC16F877A microprocessor which was manufactured by Microchip Technologies, Incorporated. The PIC board was given to each group fully assembled and no modification could be made to it. It was attached to the robot through pre-drilled holes and all wire attachments were made via the screw-down connectors. The PIC microcontroller board was designed to operate via either a 9.6VDC NiCad or Nickel Metal Hydride battery pack that will located on the robot, or a black 9VDC output, AC powered wall plug-in adapter. The board is equipped with solid state resettable fuses. If there is a short circuit or other unexpected high current situation on the board, a fuse will open to disconnect power and protect the board. The analog/digital converters in the PIC can be configured for applications other than our light sensors. Analog channel configuration jumpers are used to configure their operation. Port D has eight digital I/O pins. One of these pins must be selected and used as an output pin to turn on the FSM LED. The other seven are spare.

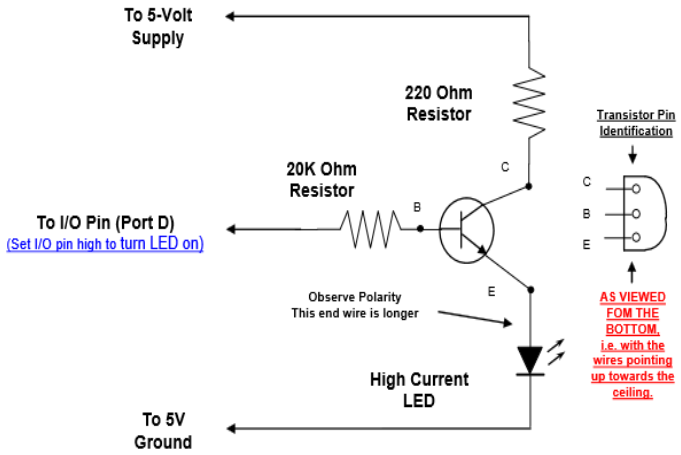


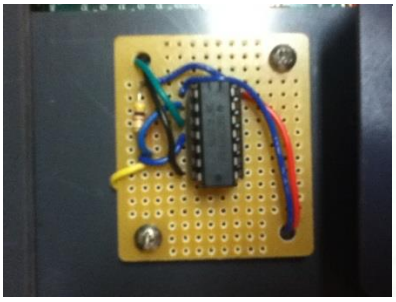
Wiring

All of the wiring was designed from scratch and both the BIS and FSM were constructed. The floor sensor module is needed to: provide a mounting position for the photo resistor and LED, as well as to provide a way for the PIC board to turn on and turn off the LED. Powering the LED on and off is accomplished by using a transistor which the pic board can send a low voltage signal to, in order to control power to the LED.

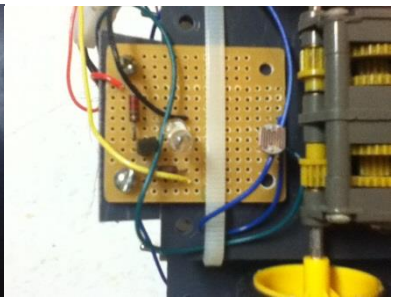
The wiring for the robot was consistent: red was always used for VCC and black for Ground. The rest of the wiring was all different colors so they could be identified easily. There was also special consideration to keeping all of the wires neat and organized inside of the robot.







Bumper Interrupt Support



Floor Sensor Module

Assembly

The assembly of the robot used primarily the hand and bench tools that were provided in the classroom. Much of the components required to make the base of the robot were constructed from the gray PVC plastic, and then screwed together. The bench tools used were the band saw, the drill press, and the sander. The hand tools used were screwdrivers, wire cutters, soldering iron/solder, etc. The process of assembling everything seemed to be simple at first, but then the group ran into a couple of design issues such as having to build around parts of the robot. These were easy fixes. Luckily, the robot assembly did not cause any problems.

Integration Testing and Evaluation

Before any integration testing could be completed, the mechanical assembly needed to be completed, and the strategy for the program figured out. Testing and integration were the two most important tasks that were performed as a group. The approach was fairly straightforward. It started with the motor speed and just having the robot go in a straight line. While a seemingly meaningless task, it is actually one of the most important tasks. Had the robot had a slightly different motor speed, it would have to be recalibrated as a different motor speed would yield different pause values when trying to preform precise tasks such as pivot 45 degrees or go forward one foot. The building block approach was important was used in precisely this manner. The robot could not progress forward to pivoting 45 degrees if the motor speed was not calibrated. The building block technique was essentially used to minimize the number of variables the robot had to deal with as the programming got more complex.

The major milestones on the Gantt chart gave the group important deadlines to meet. First, the robot needed to have a basic design. Through the alternative design matrix, multiple conceptual designs were considered, and finally a design was chosen. The next step was to construct the basic chassis and mount the motors. This involved figuring out the exact placement of all the required components and building around it. The design of the chassis was modular, so that pieces could be easily taken off to work on any component. Then, the Floor Sensor Module and Bumper Interrupt Support board needed to be soldered and configured to the robot. Once the hardware was complete, the software needed to be written and integrated with the entire system. This includes both the main loop and the bumper interrupt code. Finally, once all of these components were configured, the robot was ready for competition.

The software testing was broken up into 3 segments. First, the bottom floor light sensor was tested as it was the easiest to test because, theoretically, there were only two possible values that the floor sensor could have because it would either be reading the light off of the white side or the black side. The next segment worked on was the beacon light sensor which was one of the most difficult parts of integration testing. The reason this was so difficult was because of the strategy of the robot during the competition. The minimum strategy required the robot to determine the values of the light sensor when it got within several inches on the beacon light.

The bumpers were made more directionally sensitive as they were having trouble distinguishing between a left bumper hit and a dual bumper hit.

The sensors were calibrated using the building block method as stated above. A full list of calibration data can be found in the appendix.

The two LEDS were used for bumper testing. A program was created that would light up whenever the bumper switch was triggered. The yellow led was assigned to go off when the left bumper was hit and the red led to go off when the right bumper was hit.

Final Competition

(to be continued)

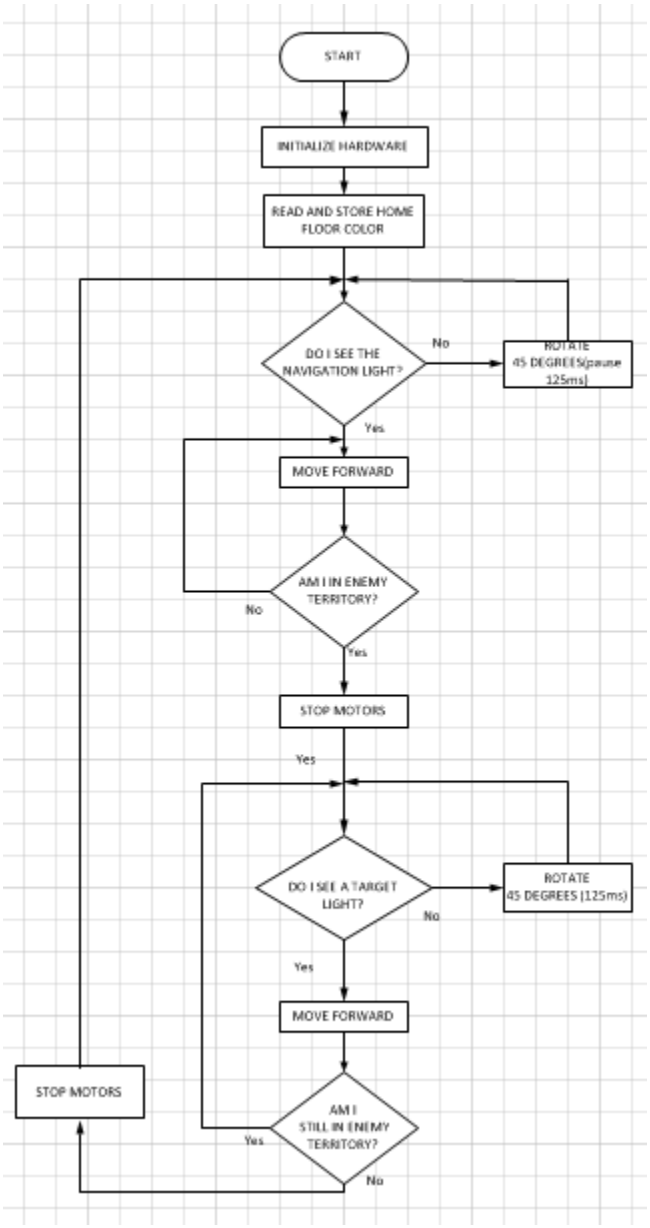
Conclusions

(to be continued)

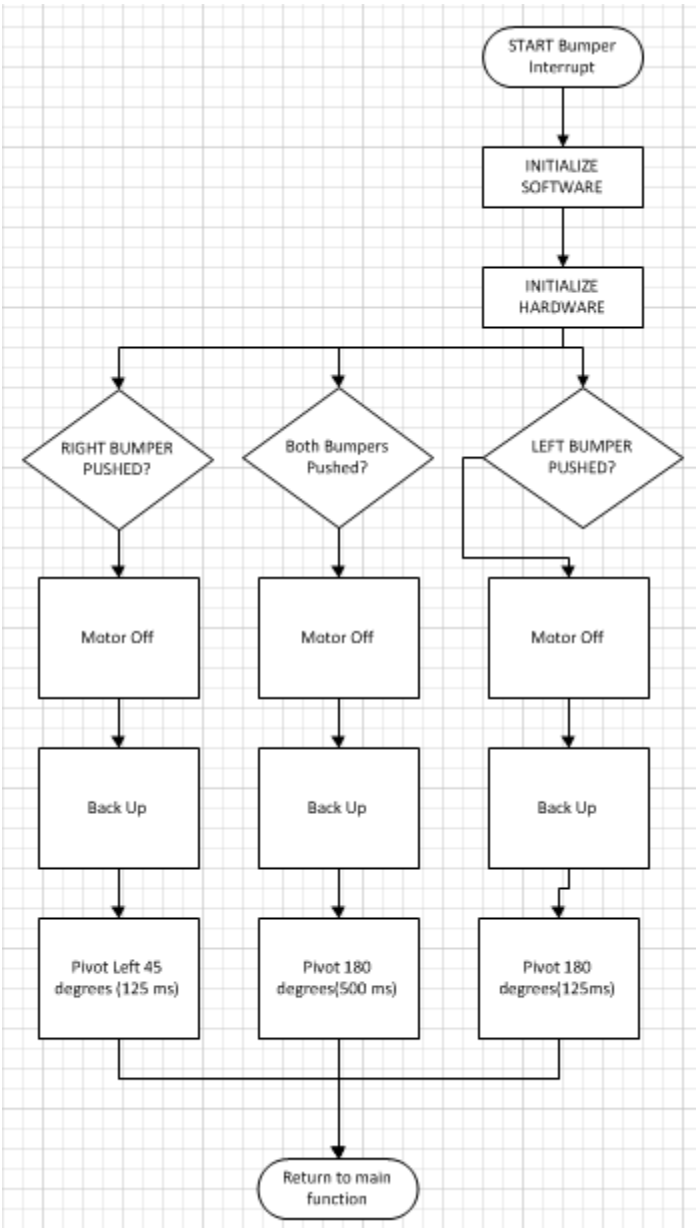
Appendix A

Code

Appendix B
Flowcharts



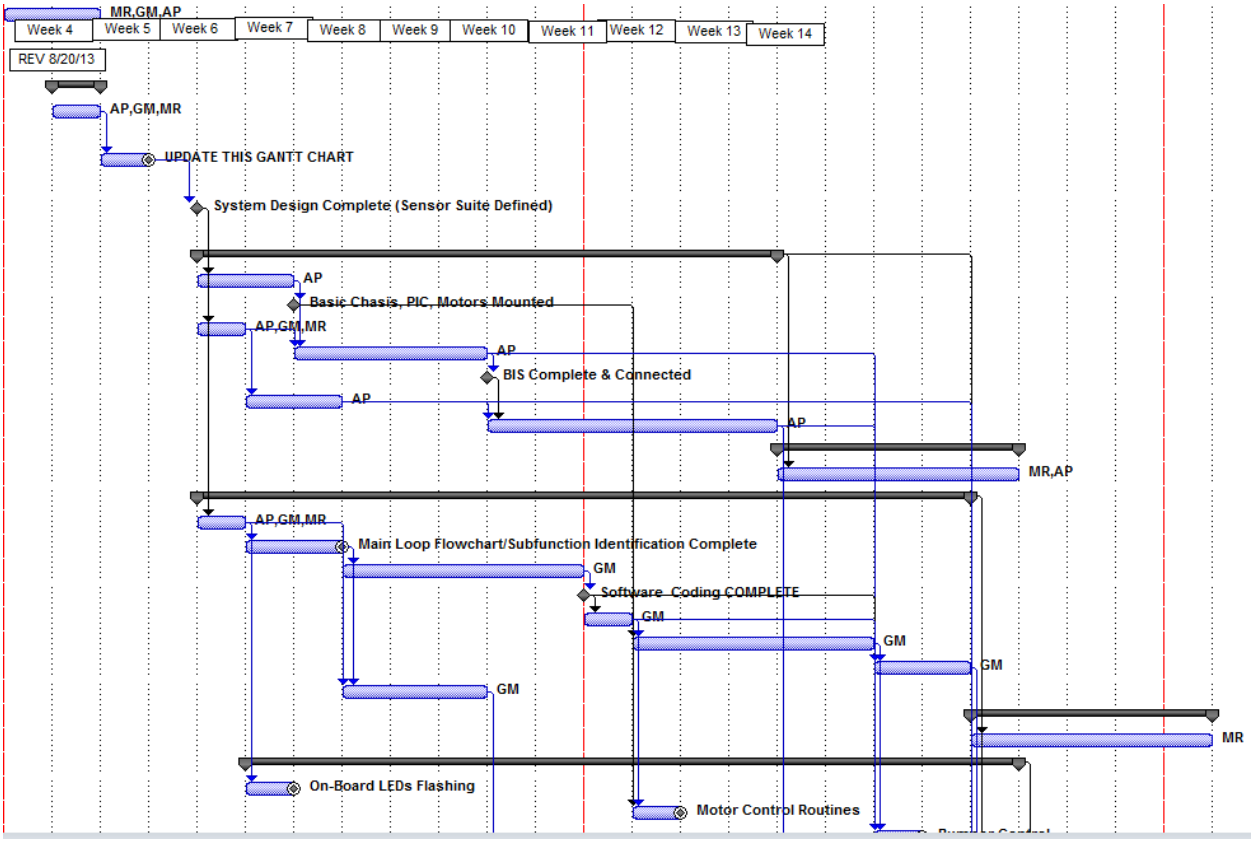
Main Flowchart



Bumper Flowchart

Appendix C

Gantt Chart



Appendix D
Work Breakdown Sheet

Major Hardware Activities-

1) Robot Chassis Construction
    a) Brainstorm Designs
    b) Create Drawings
    c) Take Measurements
    d) Cut out Chassis Pieces
    e) Use Drill Press to Create Holes in Plastic Pieces
    f) Assemble Chassis
2) Electric Work
    a) Soldering Clinic
    b) Assess Moodle for Diagrams
    c) Gather Materials (Wires, Boards, Solder)
    d) Assemble Light Senor Operations
        i) Floor Sensor Module
        ii) Navigation Light Sensor
        iii) Target Light Sensor
    e) Attach Light Sensors to the Robot Chassis
3) Bumper Interrupt Support (BIS) and Bumper Design
    a) Bumper Construction
        i) Brainstorm Bumper System
        ii) Design Bumper System
        iii) Cut out Bumpers with Machines
        iv) Use Drill Press to Create Holes
        v) Mount Bumpers to Chassis
    b) BIS
        i) Access Moodle for Diagram
        ii) Solder BIS
        iii) Connect BIS to PIC Board
        iv) Wire Pic Board to Bumper System
    c) Reinforce Bumpers

Major Software Activities-

1) Download/ Exercise SourceBoost IDE
2) Main Loop Flowchart/ Sub function Identification Complete
3) Write Main Loop Code
4) Complete Software Coding

5) Develop Motor Control Routines
6) Develop Bumper Control Routines
7) Develop Light Sensor Routines

Functionality Tests
1) On-Board LEDs Flashing
2) Motor Control Routines
3) Bumper Control
a) Calibrate Rotations and Distances
4) FSM Calibration
5) Navigation Light Calibration
6) Target Light Testing

System Integration Testing & Debugging

Logo Development
1) CNC Demonstration in Carnegie
2) Brain Storm Logo Design
3) Design Logo in Sold Works
a) Using Measurement Restrictions
4) Cut Logo
5) Mount Logo

Competition- December 4, 2013

Reports
1) Written Final Paper (MS Word)
2) Oral Presentation (MS PowerPoint)

Appendix E

Organization Chart

1. Group Member: Anthony Pacheco
    a. Role: Mechanical/Electrical Engineering
    b. Responsibilities
        i. Robot Construction
            1. Bumper design
            2. Body construction
        ii. Electrical Work
            1. Wiring – PIC board
            2. Soldering
            3. Circuit boards
2. Group Member: Greg McNeil
    a. Role: Software Programmer
    b. Responsibilities
        i. Wrote robot program
        ii. Software/Hardware Integration
3. Group Member: Miranda Rohn
    a. Role: Project Manager
    b. Responsibilities
        i. Organize the group
        ii. Write and assemble written report
        iii. Construct oral presentation
        iv. Logo design

Appendix F

General Port/Bit Assignments

PORTA – 'A' (Six I/O Pins)

Bit	I/O	Analog/Digital	Assignment	To Access
0	I	Analog	Light Sensor #1	Use "read_adc" function
1	I	Analog	Light Sensor #2	Use "read_adc" function
2	I	Analog	Light Sensor #3	Use "read_adc" function
3	I	Analog	Light Sensor #4	Use "read_adc" function
4	O	Digital	On Board Debug LED #1	Use "output_high" or "output_low" functions **0 = on; 1 = off**
5	I	Analog	Light Sensor #5	Use "read_adc" function

PORTB – 'B' (Eight I/O Pins)

Bit	Assignment
0	Unavailable for E121
1	Unavailable for E121
2	Unavailable for E121
3	Unavailable for E121
4	Unavailable for E121
5	Unavailable for E121
6	Unavailable for E121

7	Unavailable for E121

PORTC – 'C' (Eight I/O Pins)

Bit	I/O	Analog/Digital	Assignment	To Access
0	O	Digital	Motor #1 Direction Bit A **(See Table Below)**	Use "output_high" or "output_low" functions
1	O	Analog	Motor #2 Speed	Use "motorspeed" function
2	O	Analog	Motor #1 Speed	Use "motorspeed" function
3	O	Digital	Motor #2 Direction Bit A **(See Table Below)**	Use "output_high" or "output_low" functions
4	O	Digital	Motor #2 Direction Bit B **(See Table Below)**	Use "output_high" or "output_low" functions
5	O	Digital	Motor #1 Direction Bit B **(See Table Below)**	Use "output_high" or "output_low" functions
6	-	Digital	Hardware RS-232 Xmit	Transparent- Used by bootloader, putdata, putchar
7	-	Digital	Hardware RS-232 Rcv	

Motor Direction Bit Usage

Motor Direction Bit A	Motor	Result

	Direction Bit B	
0	0	Motor off
0	1	Motor On – Rotate at speed set by "motorspeed" function
1	0	Motor On - Rotate opposite direction at speed set by "motorspeed" function
1	1	Motor off **(do not use this configuration)**

PORTD – 'D' (Eight I/O Pins)

Bit	**I/O**	**Analog/Digital**	**Assignment**	**To Access**
0	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
1	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
2	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
3	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
4	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
5	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
6	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions
7	I/O	Digital	Spare I/O	Use "output_high" or "output_low" functions

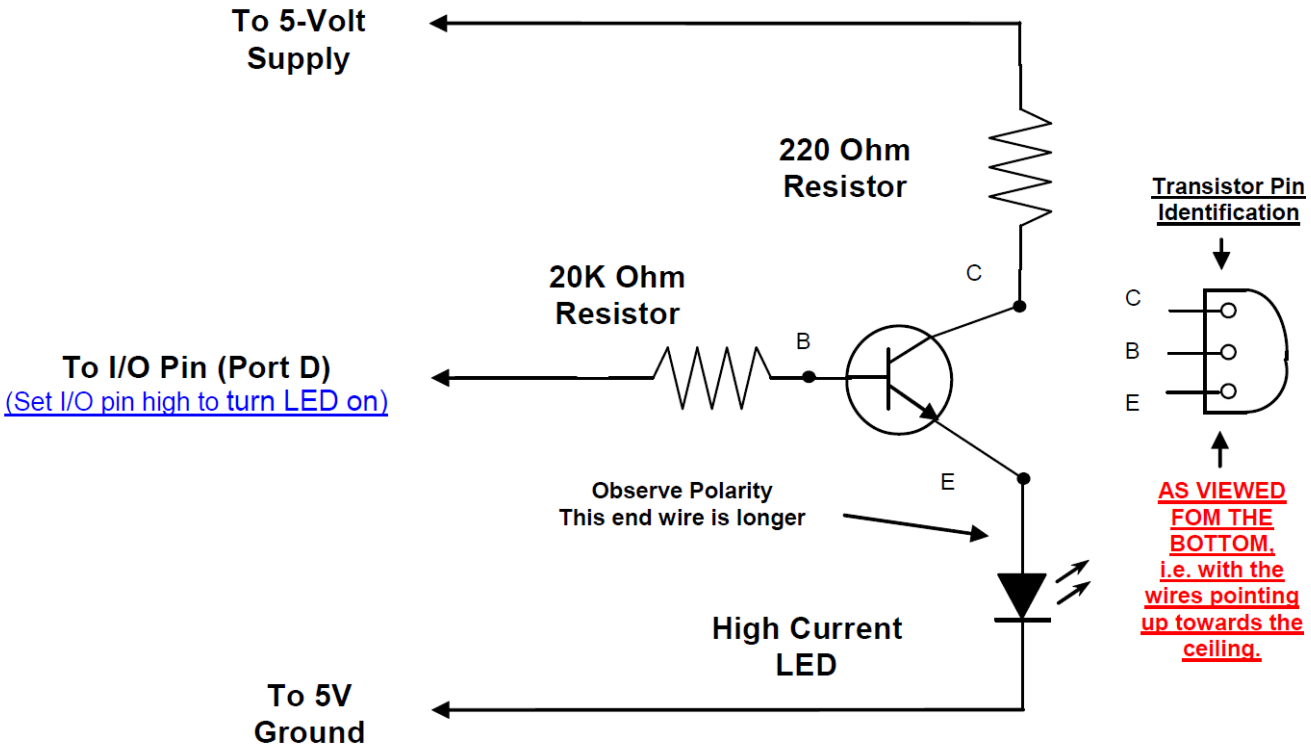
PORTE – 'E' (Three I/O Pins)

Bit	I/O	Analog/Digital	Assignment	To Access
0	I	Digital	Bumper #1	Use "read_input" function **0 = bumper closed (activated)** **1 = bumper open**
1	I	Digital	Bumper #2	Use "read_input" function **0 = bumper closed (activated)** **1 = bumper open**
2	O	Digital	On Board Debug LED #2	Use "output_high" or "output_low" functions **0 = on; 1 = off**

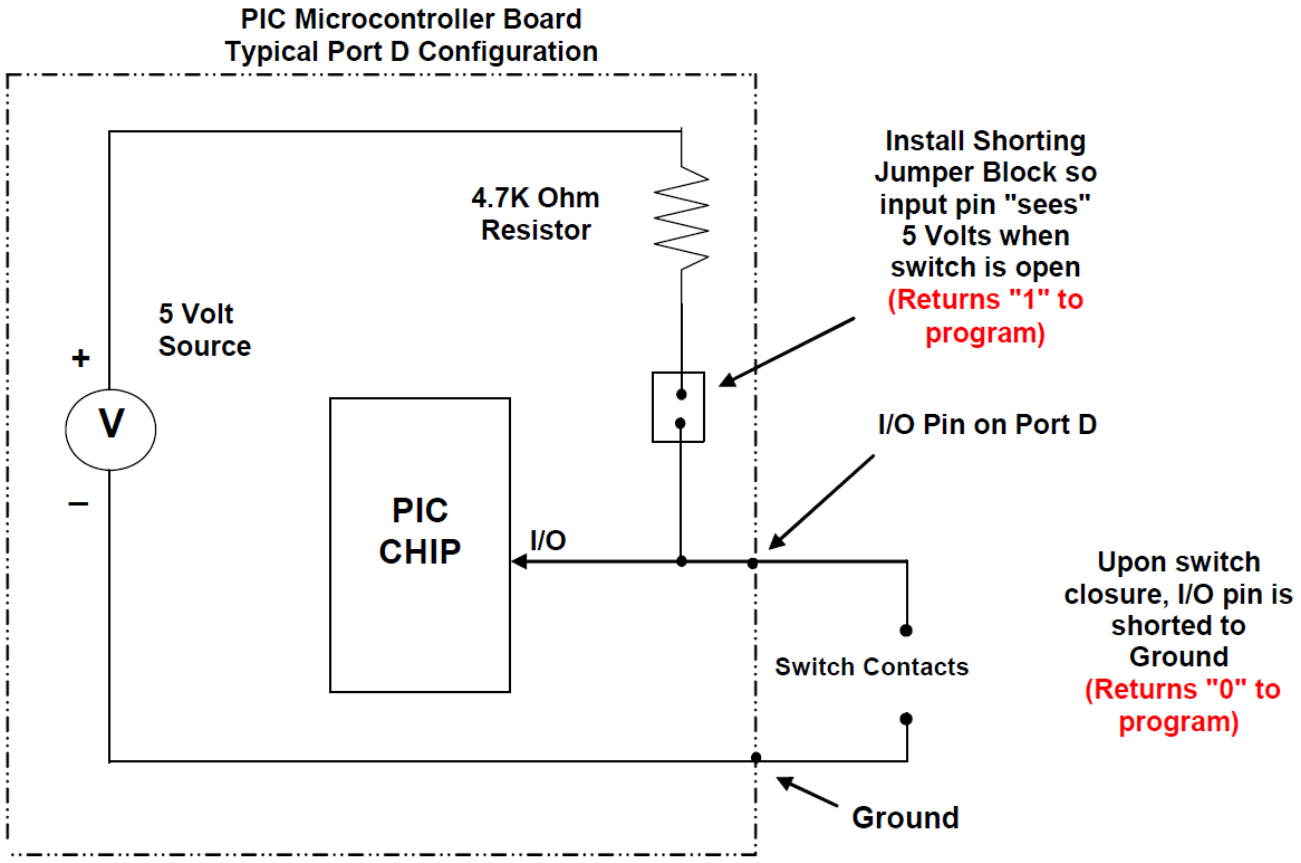
Appendix G

Electronics

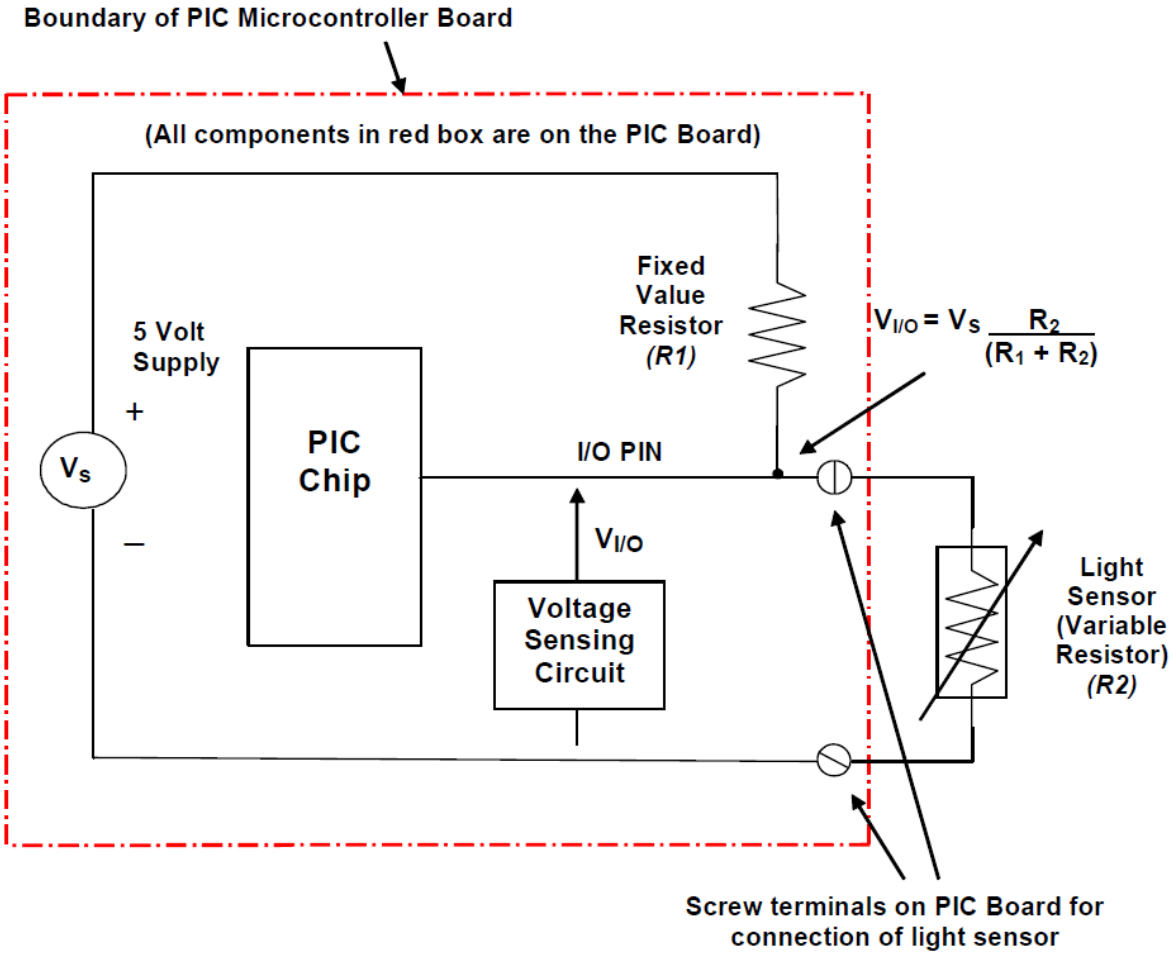
LED



PIC Board


PIC Microcontroller Board
Typical Port D Configuration
4.7K Ohm
Resistor
Install Shorting
Jumper Block so
input pin "sees"
5 Volts when
switch is open
(Returns "1" to
program)
5 Volt
Source
+
V
–
I/O Pin on Port D
PIC
CHIP
I/O
Upon switch
closure, I/O pin is
shorted to
Ground
(Returns "0" to
program)
Switch Contacts
Ground

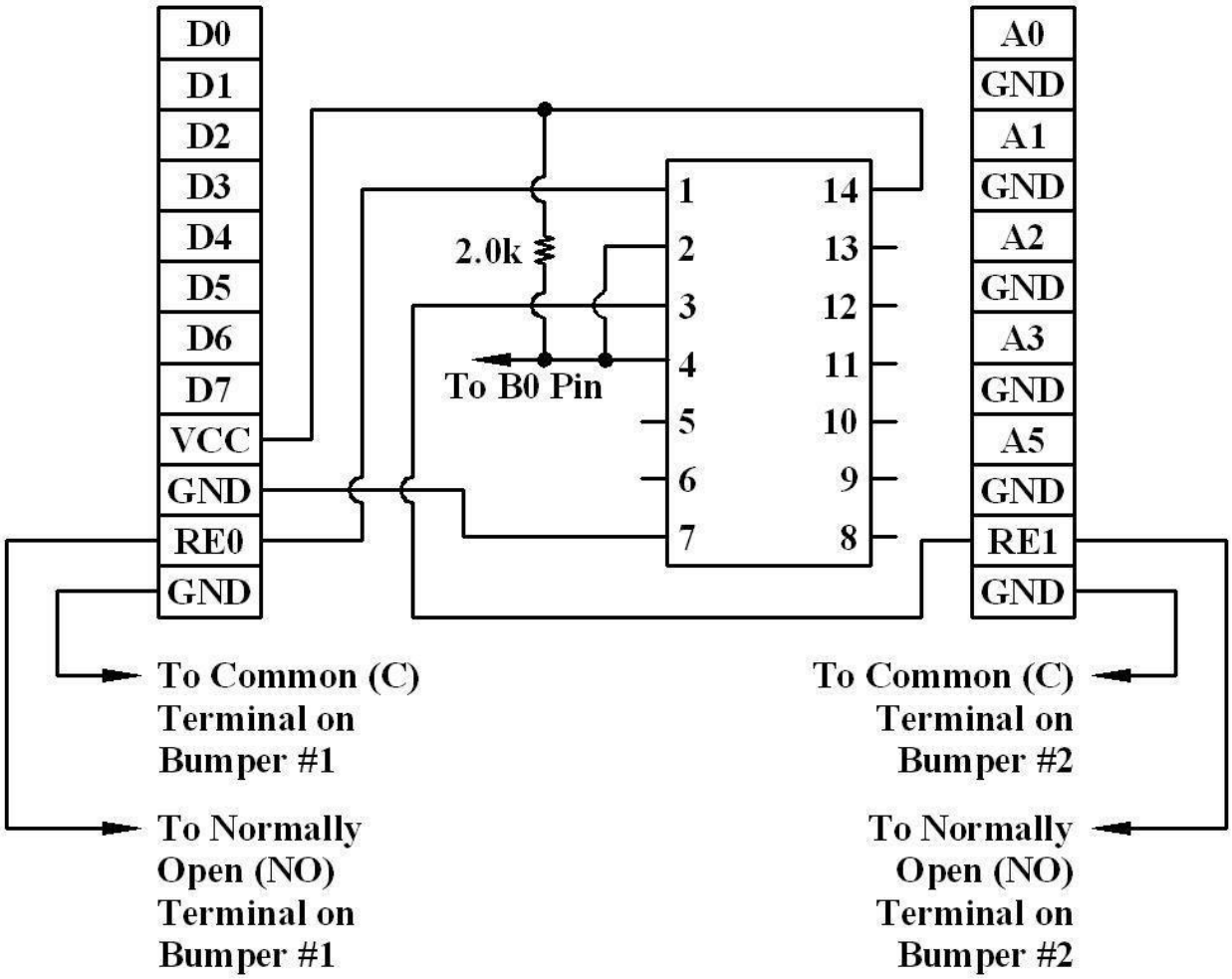
Light Sensors



Appendix H

Bumpers

Two Bumper



Appendix I

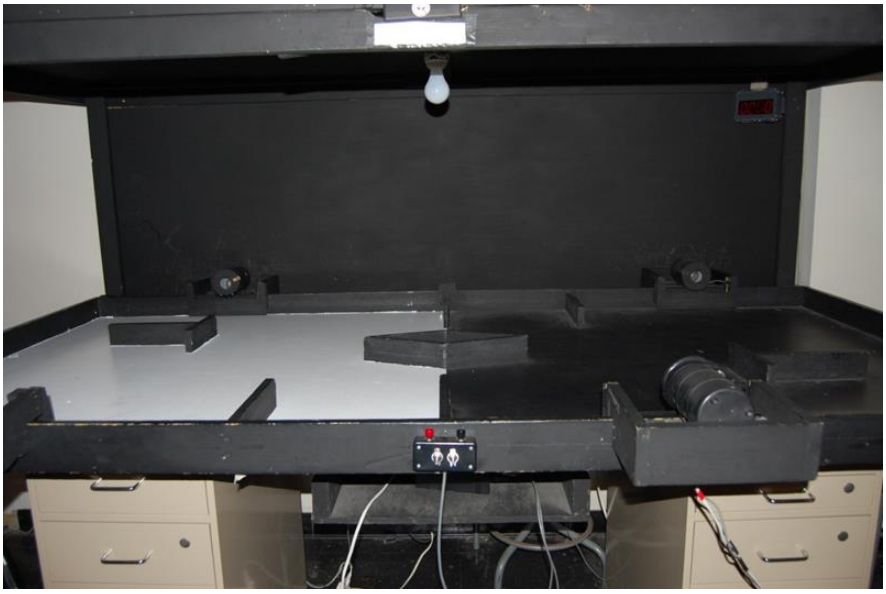
Components List

Components	**Quantity**
Top Plate	1
Bottom Plate	1
Gear Case with motors	1
Battery	1
Charger	1
AC Wall Adaptor	1
Serial Cable	1
Wheels	2
Battery Connection Cable	1
Pin Connector with Wire Lead	1
5 ½” x 5 ½” x 3/16” Grey PVC	2
5 ½” x 5 ½” x ⅛” Grey PVC	1
1” x 5 ½” x ¼”	1
Blank Circuit Board (PC Board)	1
High Current LED	1
Light Sensors	3
220 & 20k Ohm Resistor	1 ea.
2N2222 Transistor	1

½” Screws	16
Bumper Switches	2
¼” Spacers	4
¼” Screws	4
¾” Threaded Spacers	4
HEX Buffer integrated Circuit	1
14 Pin IC Socket	1
4.7kΩ Resistor	1
Electrical Tape*	lab
Tapping Screws	lab
Shrink Heat Tubing*	lab
Colored Wire	3 colors - lab
Zip Ties	2 - lab
Screwdrivers	toolbox
Wire Cutters	toolbox
Pliers	toolbox
Box Cutter	toolbox

Appendix J

Pictures



The Arena