Dhaval Patel
April 25, 2001

# LEDBOT

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

## TABLE OF CONTENTS

## ABSTRACT

The purpose of LEDBOT is to perform entertaining behaviors.  Building from a TJ Pro robot base, the additional hardware put onto the robot will increase functionality and reliability.  PIR sensors accompanied by a projectile unit will allow LEDBOT to track and shoot moving objects.  The placement of a cannon atop the TJ Pro base will provide an upward projection of a bottle cap at moving objects in its line of sight. Behavioral routines will mimic instinctual concepts such as exploring its surroundings and self-preservation.  LEDBOT is so named due to its LED display unit in front of the body that reflects its attitude.

## EXECUTIVE SUMMARY

LEDBOT is designed using a TJ Pro base unit in conjunction with various extra sensors to behave in an entertaining manner. The combination of hardware and software meshes to produce an autonomous agent that can navigate a room while avoiding obstacles in its path while also trying to detect motion through the use of PIR sensors. Another sensor unit designed specifically for the robot is the LED display.

The name LEDBOT is derived from this sensor, which reflects the attitude, or behavior, currently running. In addition to these sensors is an accompaniment of more common sensors, various actuators, and software modules known as behaviors. The more common sensors are the IR sensors used for proximity detection and ledge or wedge avoidance, bump switches that warn of contacting an object, and a voltage divider circuit which allows LEDBOT to know the current status of its batteries. All of these sensors provide a window to the environment in which LEDBOT is a part of. Reactions to various states of the sensors results in behaviors such as obstacle avoidance, which also entails avoiding ledges and wedges as well as reacting to bumps.

LEDBOT implements a fuzzy logic method to traverse its environment. This requires the use of software modules that ‘fuzzify’ and ‘defuzzify’ sensor readings. Furthermore, these modules require logical manipulation of the obstacle avoidance matrix. This provides an idea of the depth of the software written for LEDBOT. Other behaviors include initialization and calibration, checking the status of the batteries, motion detection, and capping a person.

## INTRODUCTION

The concept of an autonomous instinctual robot using information gathered from various resources will be implemented for the purposes of entertainment. The body of the robot was built from the TJ Pro model from Mekatronix, which is shown in FIGURE 1. Modifications such as the LED display reflect behavioral routines, while the projectile unit gives LEDBOT the ability to shoot moving objects within its range. The details of the project will be discussed in the following pages.

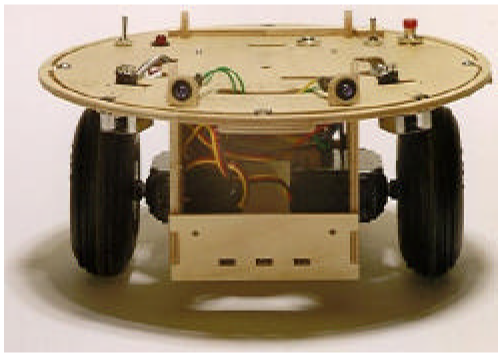


FIGURE 1: Base model of the TJ Pro.

## INTEGRATED SYSTEM

The result of integrating the various sensors and actuators to the base model of the TJ Pro produced an autonomous agent named LEDBOT. The robot uses its sensors to react to the environment and provide entertainment. The parts that comprise the system are given in the TABLE 1. The actuators, behaviors, and sensors listed in TABLE 1 provide LEDBOT the ability to sense and react according to the behaviors programmed.

| Operation Required | Implemented | Type |
|---|---|---|
| Movement | 2 Servos | Actuator |
| Weapon | 2 Servos | Actuator |
| Battery Check | 1 Software Module | Behavior |
| Behavior Reflection | 1 Software Module | Behavior |
| Calibration | 2 Software Modules | Behavior |
| Cap Firing | 1 Software Module | Behavior |
| Motion Tracking | 3 Software Modules | Behavior |
| Obstacle Avoidance | 9 Software Modules | Behavior |
| Battery Check | 1 Voltage Divider | Sensor |
| Behavioral Reflection | 1 LED Display | Sensor |
| Bump Reaction | 4 Bump Switches | Sensor |
| Ledge Wedge Detection | 1 IR Sensor | Sensor |
| Motion Tracking | 2 PIR Sensors | Sensor |
| Proximity Detection | 2 IR Sensors | Sensor |

TABLE 1: LEDBOT operations and implementation.

Integrating the total package became difficult due to inconsistencies in various sensors. The addition of the LED display became compromised sense the clock provided by the MC68HC11 processor on the MTJPRO11 microcomputer had too high of a frequency for the ALTERA CPLD chip used to implement the logic. A solution to this would be to use a 555-timer circuit that could generate an ideal square waveform with a frequency of 10 Hz.

Also the PIR sensors have a tendency to pick up background motion, or heat, while LEDBOT is moving. This was worked around via a software filter that would take the difference between the two PIR sensors. This allowed LEDBOT to track a motion that the PIR sensor read while still remaining in motion thereby preventing unnecessary stops. However, sense the PIR sensors pick up heat, the robot would occasionally move toward a nonmoving object.

Also a problem with the EOW (End-Of-the-World) sensor, which is implemented by an IR sensor placed directly in front, resulted in LEDBOT being unable to detect some ledges. This was also worked around through software. Using tolerances for detecting a ledge or a wedge prevented unnecessary back ups from false readings.

However, if LEDBOT approaches a ledge at an angle, the chances of detection lessen since there is only one EOW sensor placed directly in front of the body.

As for the integration of the actuators, a problem arises from using a servo as a motor. The servos, which were hacked to act as motors, not only provide poor connection with the HPI three-spoke chrome wheels with low profile radials, but also have varying angular velocities. This combination results in poor motion. More specifically, straight travel is unlikely without software manipulation. Also a minute hitch can be noticed due to irregularities between the wheels.

Connecting the weapon actuators also proved difficult. Since more time was required to design and mount the loading mechanism, only the motor that provided the torque to propel a bottle cap was realized. This motor, however, required extra wiring to integrate with the microcomputer; therefore it remains a non-factor due to the lack of time. Although the behavior, which activates the weapon, is already included in the software.

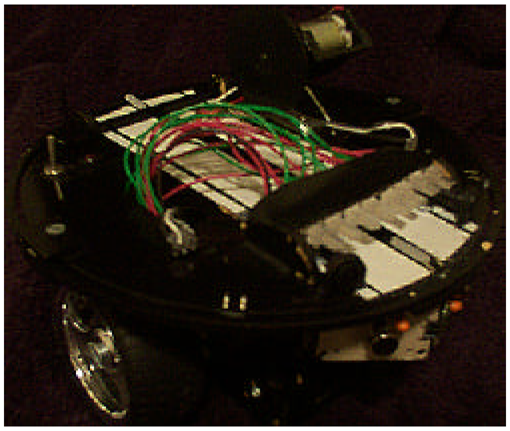


FIGURE 2: The LED Bot mobile platform.

## MOBILE PLATFORM

The concept of a circular two-wheeled one-pivot platform was the initial idea for the robot. Through research done using texts and the class website, the TJ Pro autonomous agent was remarkably similar to my idea. Since the Mekatronix TJ Pro platform was analogous to the concept that I had, the use of it as the foundation of my work lead to a stronger final result. The only problems incurred were the inability to find a small pivoting roller for the back of the robot, so the packaged plastic tailskid was used. Also the integration problems stated above remain an issue on the platform.

The provided information in TABLE 2, in regards to the mobile platform, are strictly the basic specifications found on Mekatronix's website. The height is affected by the addition of the servos used for the weapon system as well as the weight due to the addition of extra sensors and actuators.

| TJ Pro Specifications | |
|---|---|
| Dimensions | 17.5 cm diameter x 7.5 cm tall |
| Weight | 509.6 g |
| Speed | 0.225 m/s |
| Power Source | 6 AA NiCd Cells |
| Drive System | 2 wheel differential drive at geometric center |
| Processor | Motorola MC68HC11 @ 2 MHz with 32k SRAM |

Table 2: Base model specifications.

The additions made on the base platform include replacement of the tires provided with HPI's low profile radials and three-spoked chrome wheels. The low battery indicator which would initiate a recharge behavior, the LED display in front of the module to indicate the current behavior routine, the PIR sensors used for motion detection, and the EOW sensor for avoiding ledges and wedges have all been added to the system. The addition of projectile unit completes the platform.

Problems for each sensor and there relative solutions are included in the following pages. LEDBOT, although not completely integrated, does perform the tasks set during the initial stages of conception. These include obstacle avoidance and motion tracking with complex software routines.

## ACTUATION

A pair of servos, hacked as gearhead motors, drives the robot. As shown in Table 1, the platform requires six AA NiCd cells providing seven to eight volts. The projectile unit requires an actuator that will trigger the process and also one that propels the projectile. The addition of extra actuators adds complexity and weight to the platform and has proven difficult to integrate within the time frame.

### DRIVE MOTORS

The hacked servos used for the motors required software manipulation to perform decently. The introduction of a lag factor was experimented with to allow for straighter traversal. Although effective, especially for maximum speeds, it was not implemented since a perfectly straight motion was unnecessary for LEDBOT.

Also a smoothing function, which can be found in the IC Primer handout, was used to allow for a cleaner motion. Since the obstacle avoidance behavior implemented fuzzy logic, changes in speed were frequent and sometimes abrupt. The smoothing algorithm would gradually make the change from the previous velocity to the required velocity. The formula seen below was used to realize this idea. Ultimately, through experimentation, 90% of the new velocity combined with 10% of the previous velocity proved to be the most effective.

speed = ((previous velocity) + (9 * speed)) / 10

The code employing this algorithm can be found in Appendix B under the move routine. The method prevented the motors from unnecessary hitches in motion.

Traversal of environments proved to be more stable with the combination of fuzzy logic and velocity smoothing.

### WEAPON SYSTEM

The projectile unit is a combination of a common disc shooter and a loading mechanism. The loading device was not designed and therefore not mounted, however, the disc shooter did provide a motor that would torque a projectile. Since the initial idea was to shoot bottle caps, the motor in the disc shooter was removed and mounted onto the platform. However problems arose when trying to activate the motor. The additional circuitry that would be required to simply activate the motor is shown in FIGURE 3.

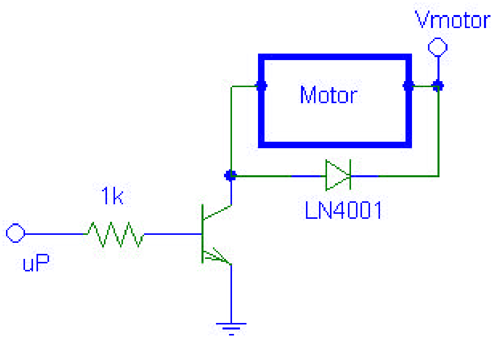


FIGURE 3: Weapon motor circuit.

The additions required provide the motor with adequate current to initialize. However, the time required to compose the circuit and mount the device in conjunction with the additional hardware for the weapons system has prevented the completion of the projectile system.

## SENSORS

The use of the sensors will allow the robot to interact with its surroundings. Enabling behavioral routines that will give the robot characteristics. The TJ Pro comes with the sensor package given in Table 2.

| Quantity | Type | Function |
|---|---|---|
| 2 | IR Emitter | Emits a wavelength of 940 nm |
| 2 | IR Detector | Provides analog channel readings |
| 4 | Tactile Switch | Bump detection |

Table 3: Sensors and their functions for the base TJ Pro model.

The additional sensors, which have been discussed in previous sections, will allow LEDBOT to interact with its environment in an entertaining manner. The sensors discussed in this section comprise the total package.

## BATTERY SENSOR

LEDBOT has the ability to detect the level of charge it has remaining in the six AA NiCd batteries. By using a voltage divider circuit the voltage level can be monitored through an analog port. If the charge falls below a set tolerance, a flag is set and the system goes into standby mode. More detail about the behavior can be seen in the behavior routine. This section will focus more on the layout.

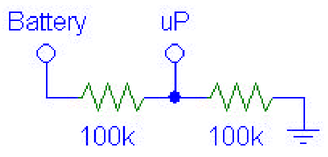


FIGURE 4: Voltage divider.

The use of 100 kΩ resistors prevents overdrawing of the batteries and thus avoiding drainage of the charge through a circuit that is meant only to check it. The layout of the design shown in FIGURE 4 provides the details of the circuit. Notice that the sum voltage from the batteries is being checked and not a voltage reading from the microcomputer. This makes sure a drop can be noticed before the charge falls below five volts, which is the regulated amount for the system. A detailed description of how the circuit is read can be found under the behaviors section.

## BUMP SWITCHES

The bump switches are included in the TJ Pro platform and therefore are easily integrated. Comprised of a multiple voltage dividing circuit layout, the four bump sensors can be read through one analog port. Each bump sensor, when activated, results in a unique voltage reading. This can be interpreted by the software in various ways to react to the object that was just bumped.

| Placement | Part Number | Input Reading |
|---|---|---|
| Front | FBCSW | 126-130 |
| Left | FBLSW | 77-81 |
| Right | FBRSW | 43-46 |
| Back | RBSW | 17-24 |

TABLE 4: Bump switch specifications.

The readings shown in TABLE 4 were used to program the bump detection routine, as well as the initialization routine.

## EOW

Using an IR detector in conjunction with a IR emitter allows LEDBOT to detect and avoid ledges and possible places were it could get stuck, or wedges. The detector was placed in front of the module facing downwards and slightly outwards. The emitter was located behind the front faceplate of the platform and was also rapped with electrical tape to prevent leakage and false readings.

Although, highly effective method in implementing an EOW sensor, problems arise since only one IR detector is used. The placement prevents LEDBOT from seeing a ledge in time if approached from an angle, although wedges seem less of a problem. Tolerances were set for the sensor for both ledge and wedge detection. More details can be found in the behavior section.

## IR SENSORS

The two IR detectors and emitters used for proximity detection of objects were part of the TJ Pro package. This proved easy to integrate into the system, and complex software routines were used to perfect these sensors. Along with software smoothing and fuzzy logic manipulation, a correction factor was implemented for the IR sensors. Through experimentation, a noticeable performance hitch was noticed between the detector pairs. The graph given in FIGURE 5 shows this difference clearly, especially when detecting farther objects or no objects at 40 cm to 60 cm.

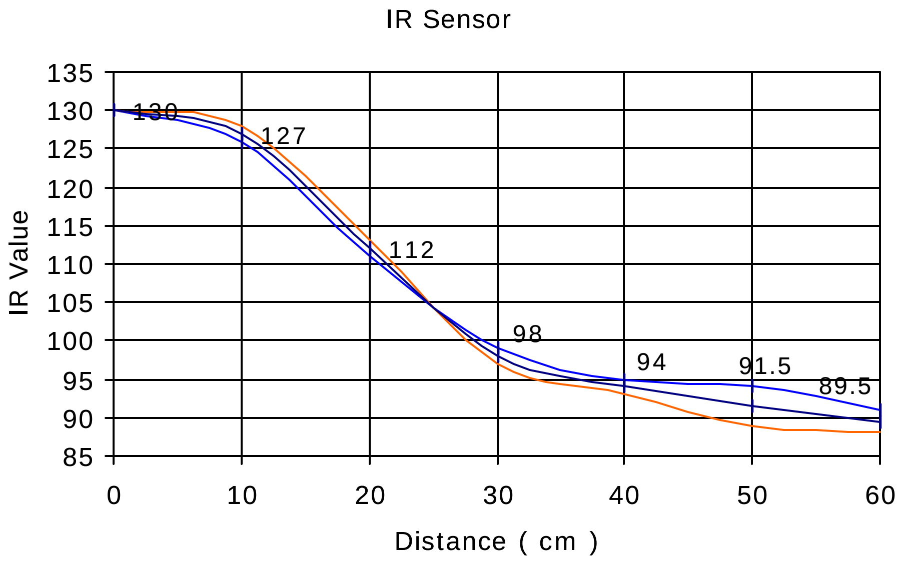


FIGURE 5: Output graph of IR sensors.

The graph shows the left IR in orange and the right IR in blue. The middle line with the values next to each significant distance marker is the average value of the sensor. These averages were used in the calibration technique discussed later.

## LED DISPLAY

The behavioral reflection concept began as an idea to use an array of LED's to tell what state the robot is currently running. Knowing that I wanted the LED display to represent four states, I designed the display to emit two colors in two forms. The first being a flashing state and the other would cause the LED's to run back and forth. A logic table is given in Appendix A, along with the equations derived from it.

The behavioral LED display was then proven using Logic Works. The schematic was then transferred into Max+Plus II, which allowed me to compile the diagram into VHDL and download the logic into a single chip. Without converting the logic into VHDL, approximately 25 integrated circuit chips would have been used to implement the process. The logic design used can be seen in Appendix A.

Using tri-mode LED's, the color of the display can be changed from green to red using the LIGHT switch shown in the diagram given in Appendix A. Also, the mode of the display can be triggered to run back and forth or to flash. This allows for four states of operation shown in Table 5.

| Mode | Behavior | Description |
|---|---|---|
| 1 | Initialization | Flashing Green |
| 2 | Wandering | Running Green |
| 3 | Tracking | Flashing Red |
| 4 | Low Battery | Runing Red |

TABLE 5: LED display behavior reflection modes.

To clarify the modes of operation, flashing means that all eight LED's that make up the array turn on and off. Running means that the LED's turn on and off one at a time to give the appearance of running back and forth.

An ALTERA CPLD was used to realize the design and a test was conducted on its operation. The results showed that the design does prove effective, however, the CPLD used demonstrated the most promising outputs using a frequency of less than 20 Hz, ideally 10 Hz. Obviously the 2 MHz or the 40 kHz provided by the microcomputer would not be effective. The solution to this problem was the use of a 555-timer circuit. However, due to the lack of time, this circuit was not constructed and the clock for the device was left floating. Resulting in a system that is unstable but still effective. The display still remains connected in every other aspect. The three controlling switches were connected on the first three bits of Port D, or the SPI Port. The programming details are shown in the appropriate behavioral routine.

### PIR SENSORS

The motion detectors are actually PIR sensors. They detect motion by sensing heat variations in its line of sight. This is where some errors can enter the method of detection of a person. Stray heat sources in a room can cause the PIR sensors to read a positive signal when in reality there is nobody moving in front of LEDBOT. Also, for LEDBOT to detect motion while moving also led to stray noise in the readings. The sensors are mechanically nonadjustable; therefore a software solution was devised.

In order for the detection of motion to take place while LEDBOT moves around its environment, a software fix was required. Through various test procedures that read the output of the two PIR sensors mounted in the front of the module, a pattern was noticed. While in motion, the PIR sensors tend to read identically, whereas a movement, beside itself, would reflect in a difference in the outputs. Using this information, a motion detection behavior was coded. More detail is given in the behavior section. As for the wiring, the two devices were directly connected into analog ports.

## BEHAVIORS

Programming the robot to model entertaining behaviors required various routines that mimicked pet-like mannerisms. The combination of multitasking and fuzzy logic basics provided an excellent basis for the goals desired. The location of a warm body would signal a firing routine. Once a target is located the robot proceeds to follow the object and fire a projectile. If the battery becomes dangerously low, LEDBOT will signal for help via the LED display and remain in a dormant state until it is recharged. This behavior can be modified to do various reactions. The robot may go to a homing beacon or it can be programmed to follow a person until that person recharges him. All the behaviors are detailed in the following section.

| Behaviors | Routines Required |
|---|---|
| Battery Check | void batteryCheck(void) |
| Calibration | void init(void)<br>void calibrate(void) |
| Cap Firing | void capU(void) |
| Behavior Reflection | void ledDisplay(int) |
| Motion Tracking | void motionDetect(void)<br>void move(int)<br>void turn(int) |
| Obstacle Avoidance | void obstacleAvoid(void)<br>void avoidLedges(void)<br>void bumpDetect(void)<br>void fuzzify(int)<br>void defuzzify(void)<br>int fuzAnd(int, int)<br>int fuzOr(int, int)<br>void move(int)<br>void turn(int) |

TABLE 6: Various routines required to run a behavior.

The listing shown in TABLE 6 is an outline of the behaviors implemented by LEDBOT as well as the information to follow.

## BATTERY CHECK

By accessing the battery sensor through an analog port, LEDBOT can keep track of its power level. If the system falls below a set tolerance, given by the constant 'TOLERANCEBATTERY', LEDBOT is currently programmed to spin in a circle and wait for someone to recharge him.

The only remaining task for LEDBOT after detecting a low voltage level is to change the mode of the LED display to that of a low battery setting.

## CALIBRATION

This routine begins with the initialization of the program. All global variables are zeroed to make sure no stray values bleed through the code. The ports, motors, and clock are all initialized, Port D is set for output and the LED display is set to initialization mode, the IR emitters are turned on, and the obstacle avoidance matrix is created. Then LEDBOT waits for a tap on the back bumper for the calibration to take place.

Before calibrating, LEDBOT must be placed close to a wall, from which it will move back or forward to get an appropriate reading from the IR sensors. An appropriate reading is one that provides a distribution between the fuzzy logic values for the IR inputs. More information on this is given in the obstacle avoidance behavior.

The EOW sensor is also calibrated during this procedure along with the IR sensors. All of the IR detector inputs are smoothed by taking in 20 readings and averaging them to produce a more accurate result. This reduces the effects of false readings and provides greater accuracy in calibration.

## CAP FIRING

This behavior occurs only when the PIR sensors are tracking a moving object. If the left PIR sensor flags a motion, it waits for a flag from the right to detect some motion as well. This results by LEDBOT tracking a moving object in one direction, and then noticing a motion in the opposite, implying a person in the middle. The accuracy of this routine is questionable, since testing has been difficult without the firing mechanism in place and a floating clock for the LED display outputting questionable mode settings.

## BEHAVIORAL REFLECTION

The LED display is manipulated via the first three pins of the SPI port. Using the port as an output, the three pins can be controlled to act like switches for the three inputs required for the CPLD. The format of the SPI port can be seen in TABLE 7, along with the associated behaviors.

| Mode | Behavior | Description | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Initialization | Flashing Green | 0 | 1 | 0 | X | X | X | X | X | X |
| 2 | Wandering | Running Green | 0 | 0 | 0 | X | X | X | X | X | X |
| 3 | Tracking | Flashing Red | 1 | 1 | 0 | X | X | X | X | X | X |
| 4 | Low Battery | Runing Red | 1 | 0 | 0 | X | X | X | X | X | X |

TABLE 7: SPI port manipulation for the LED display.

Bits 0, 1, and 2 are the LIGHT, MODE, and ENABLE switches respectively in the Max+Plus II schematic. Notice that a compilation error would occur with ENABLE and MODE being labeled inconsistently. In the schematic that was compiled, ENABLE and MODE were relabeled to qa and qb respectively. Also, notice the switches are active low.

## MOTION TRACKING

The difference taken of the two inputs of the PIR sensors provides an accurate assessment of the motion, or heat, around LEDBOT. Although stray detections will result in inappropriate movements, they are unavoidable with the low cost equipment used. The range of the sensors are reported to be three meters, they test closer to one or one and a half meters when using this motion detection algorithm. The tolerance for the PIR sensors can be adjusted to be more sensitive, but this simply results in more inappropriate movements.

Once a reading is shown to have a difference within the set tolerance, LEDBOT has been programmed to move toward that direction by using the turn and move routines. Turning to a certain degree in either direction was tested by a code provided in the introduction of the code, which is given in Appendix B.

Through experimentation, it was found that the required speed to turn in the right direction in certain degree was much greater than the required speed to turn left. The values are set in the global constants ‘TURNLEFT’ and ‘TURNRIGHT’. They can be manipulated in conjunction with the wait function following the turn commands to align LEDBOT to varying degrees. However, since only two PIR sensors are used, only a certain degree of accuracy can be maintained in following a movement.

## OBSTACLE AVOIDANCE

The final behavior could have been implemented using very simple coding. However, the results would have proved to be too rough to appear life-like. This is why fuzzy logic coding was used to implement this behavior. Following the notes on fuzzy logic provided by Kevin Harrelson, the code was constructed using various modules.

First the IR sensors are read by smoothing 20 readings. The left and right IR readings are then sent to a module named ‘fuzzify’, which separates the one value for each sensor into percentages of five states of distance. These include very near, near, far, very far, and nothing. These distance values are initially set during the calibration phase to fall between the constant values of very near, which is 127, and nothing, which is 88. These set values can be changed via the global constants ‘VNEAR’ and ‘NOTHING’; however, they were experimentally obtained and adjusted to work with the IR emitters and detectors on the platform.

After the inputs are ‘fuzzified’, the routine ‘defuzzifies’ the results by logically ‘and-ing’ the inputs and ‘or-ing’ the result with the obstacle avoidance matrix created in the initialization phase.

| | R | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| L | IR | VN | N | F | VF | N |
| 0 | VN | HR | HR | HR | HR | HR |
| 1 | N | HL | L | HR | R | R |
| 2 | F | HL | HL | S | S | S |
| 3 | VF | HL | L | S | S | S |
| 4 | N | HL | L | S | S | S |

TABLE 8: Obstacle avoidance matrix.

The obstacle avoidance matrix shown in TABLE 8 is what is used to logically ‘or’ the results discussed above. The matrix values HL, L, S, HR, and R represent motor values for hard left, left, straight, hard right, and right respectively. The left of the matrix is the left IR percentages while the top are the right values. These are represented by VN, N, F, VF, and N, which stand for very near, near, far, very far, and nothing. The values are held after ‘fuzzification’ in a global array called ‘irFuzzyLeft’ and ‘irFuzzyRight’ which can be noticed by the numbering from zero to four. Once the values are logically ‘or-ed’, the weighted-average output is calculated. This value is passed on to the move routine, which treats the number as a velocity for the appropriate motor.

This behavior proved to be very effective in traversing an environment smoothly. However, combining this routine with the motion detection routine, results in some hindrances.

## CONCLUSION

The design has gone through many modifications due to the limiting factors of time, money, and experience. The task of integration proved to be the most difficult aspect of building LEDBOT. I have provided my results as complete as possible with appropriate steps in correcting my errors. Although integration of the actuators, behaviors, and sensors was less than perfect, I believe that LEDBOT performs 90% of the tasks it was set out to do.

Even though some of the sensors are not as stable as I would have preferred, they still perform well enough to get an idea of what is going on. All of the design goals were met except for the weapon system, which remains incomplete due to time constraints and a lack of experience. The LED display worked during testing at 10 Hz with an oscilloscope, however, integration with the MTJPRO11 board resulted in a less than stellar performance.

The EOW and PIR sensors worked well, although improvements could be made with more money and time. The software seams to be the best accomplishment for this project. With the culmination of lectured concepts and previous work, the program implemented everything from simple smoothing techniques to complex fuzzy logic coding in an attempt to get the best performance out of relatively inexpensive and inaccurate devices. LEDBOT remains hopeful for a brighter, more prosperous future.

## DOCUMENTATION

A Antonio Arroyo, “Intelligent Machines Design Laboratory”, Target Copy, Copyright 1992 by Fred G. Martin.

A Antonio Arroyo, “MIL – IC Primer”, Handout from associated professors in Electrical Engineering, University of Florida.

A Antonio Arroyo, “MIL – ICC11 Primer”, Handout from associated professors in Electrical Engineering, University of Florida.

Keith L. Doty, Reid Harrison, “Robot Programming”, Handout from EEL 5934, University of Florida, November 2, 1994.

Keith L. Doty, “TJ Pro Assembly Manual”, Copyright 1999 Mekatronix.

Keith L. Doty, “TJ Pro Users Manual”, Copyright 1999 Mekatronix.

## APPENDIX A

The method used to implement the LED display is detailed in this section. First the logical table required to derive the equations used was made. This is shown in TABLE 9 below.

| qa | qb | qc | q0 | q1 | q2 | q3 | q4 | q5 | q6 | q7 | dc | d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

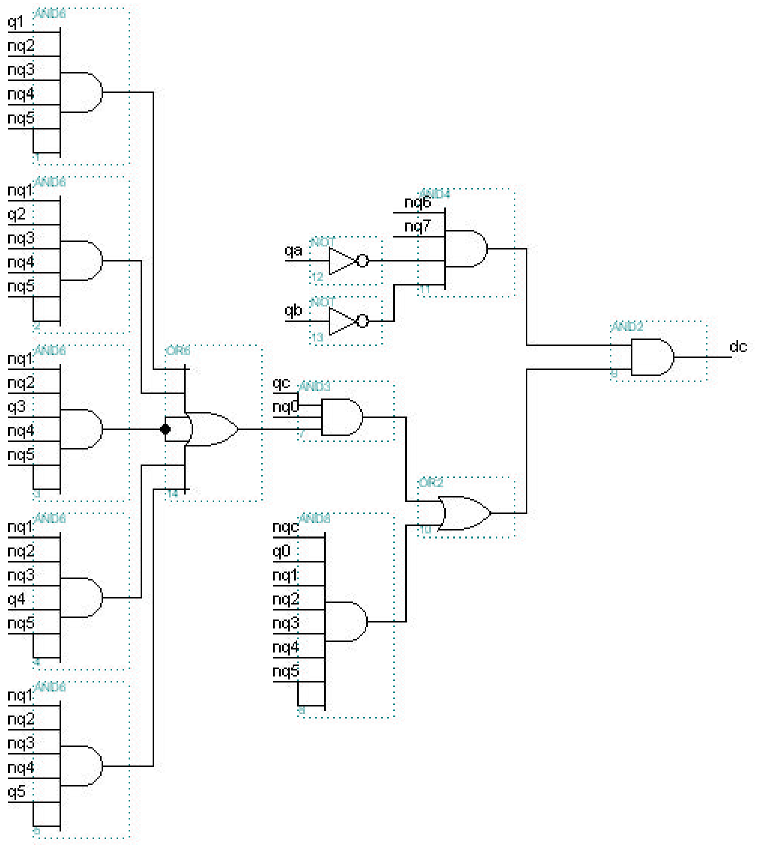
Table 9

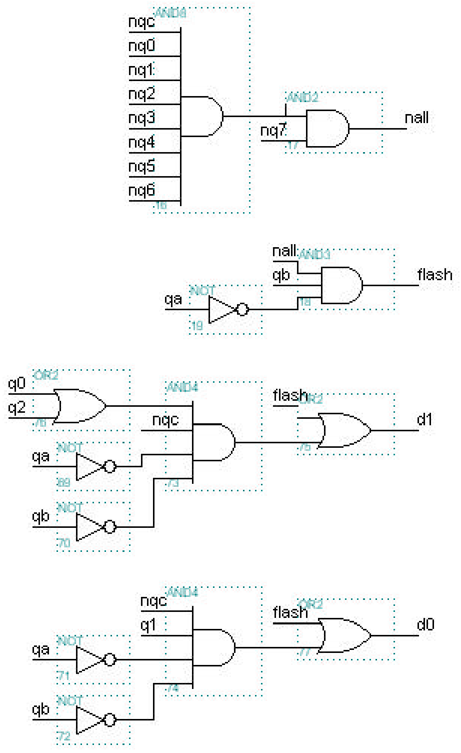
Notice that the control bits qa and qb refer to ENABLE and MODE. The qc bit is simply a control bit used for the logic, it does not refer to the LIGHT switch shown in the Max+Plus II schematic.

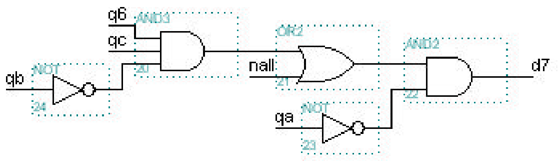
The equations derived from the logic table are given below in simplified form.

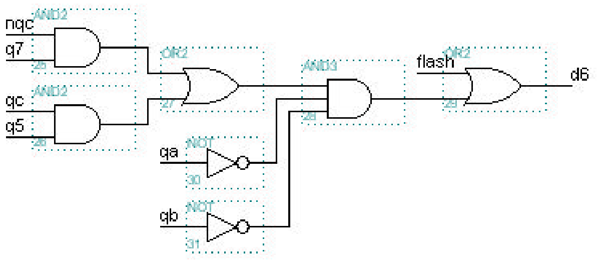
```
dc = /qa /qb /q6 /q7  [  /qc  q0  /q1 /q2 /q3 /q4 /q5
                       +  qc  /q0  (  q1 /q2 /q3 /q4 /q5
                                   + /q1  q2 /q3 /q4 /q5
                                   + /q1 /q2  q3 /q4 /q5
                                   + /q1 /q2 /q3  q4 /q5
                                   + /q1 /q2 /q3 /q4  q5  )  ]

d0 = /qa /qb /qc  q1  +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7 )

d1 = /qa /qb /qc  (  q0  +  q2  )
      +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  )

d2 = /qa /qb  (  /qc  q3  +  qc  q1  )
      +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  )

d3 = /qa /qb  (  /qc  q4  +  qc  q2  )
      +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  )

d4 = /qa /qb  (  /qc  q5  +  qc  q3  )
      +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  )

d5 = /qa /qb  (  /qc  q6  +  qc  q4  )
      +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  )

d6 = /qa /qb  (  /qc  q7  +  qc  q5  )
      +  /qa  qb  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  )

d7 = /qa  (  /qc /q0 /q1 /q2 /q3 /q4 /q5 /q6 /q7  +  /qb  qc  q6  )
```

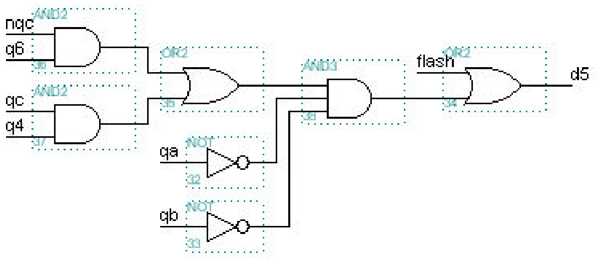
The systems of equations were then logically implemented using Logic Works. This proved effective, so the schematic was transferred to Max+Plus II so it could be compiled and transferred to an ALTERA CPLD chip. The schematics are given in the pages that follow.

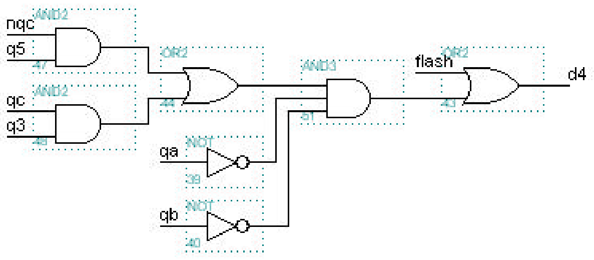

q1
nq2
nq3
nq4
nq5
nq1
q2
nq3
nq4
nq5
nq1
nq2
q3
nq4
nq5
nq1
nq2
nq3
q4
nq5
nq1
nq2
nq3
nq4
q5
nq6
nq7
qa
qb
qc
nq0
nqc
q0
nq1
nq2
nq3
nq4
nq5
dc

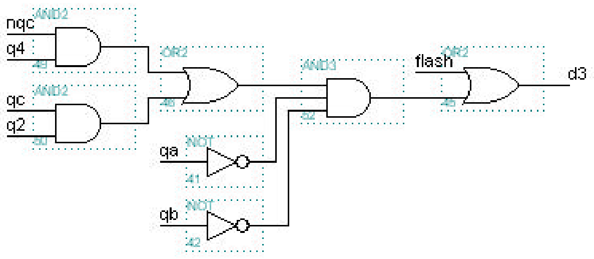

nqc
nq0
nq1
nq2
nq3
nq4
nq5
nq6
nq7
nall
nall
qb
qa
flash
q0
q2
nqc
qa
qb
flash
d1
nqc
q1
qa
qb
flash
d0

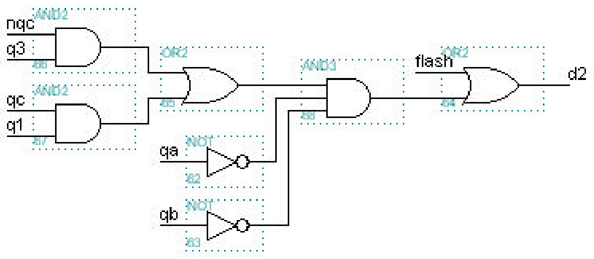

q6
qc
qb
nall
qa
AND3
OR2
AND2
NOT
NOT
24
20
21
22
23
d7

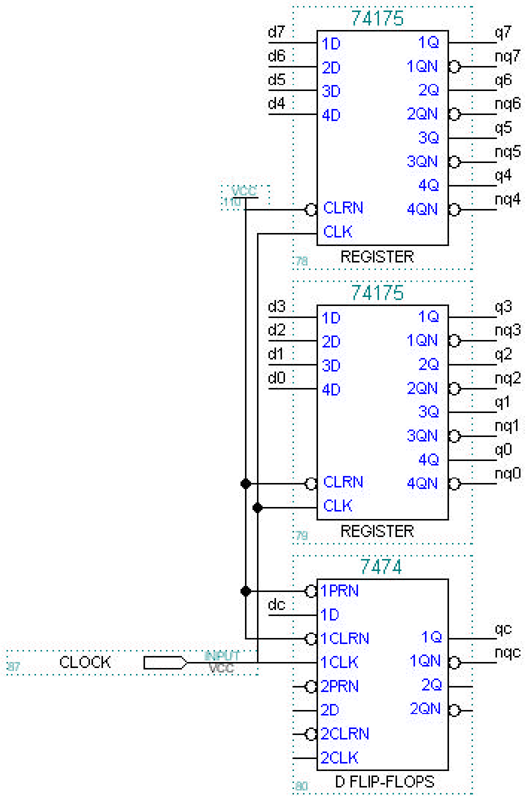

nqc
q7
qc
q5
qa
qb
flash
AND2
AND2
OR2
AND3
OR2
NOT
NOT
25
26
27
28
29
30
31
d6

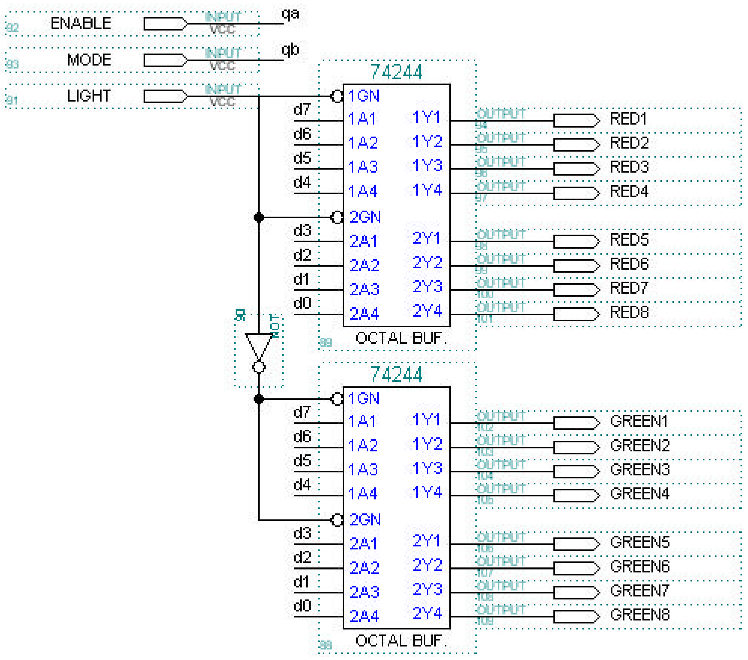

nqc
q6
qc
q4
qa
qb
flash
AND2
AND2
OR2
AND3
OR2
NOT
NOT
36
37
35
38
34
32
33
d5


nqc
q5
qc
q3
qa
qb
flash
AND2
AND2
OR2
AND3
OR2
NOT
NOT
47
48
44
51
43
39
40
d4


nqc
q4
qc
q2
qa
qb
flash
AND2
AND2
OR2
AND3
OR2
NOT
NOT
49
50
46
52
45
41
42
d3


nqc
q3
qc
q1
qa
qb
flash
AND2
AND2
OR2
AND3
OR2
NOT
NOT
66
67
65
68
64
62
63
d2


74175
1D
2D
3D
4D
1Q
1QN
2Q
2QN
3Q
3QN
4Q
4QN
CLRN
CLK
REGISTER
d7 d6 d5 d4
q7 nq7 q6 nq6 q5 nq5 q4 nq4
VCC
74175
REGISTER
d3 d2 d1 d0
q3 nq3 q2 nq2 q1 nq1 q0 nq0
7474
1PRN
1D
1CLRN
1CLK
2PRN
2D
2CLRN
2CLK
1Q
1QN
2Q
2QN
dc
qc
nqc
D FLIP-FLOPS
CLOCK
INPUT
VCC


ENABLE
MODE
LIGHT
INPUT
VCC
qa
qb
74244
1GN
1A1
1A2
1A3
1A4
2GN
2A1
2A2
2A3
2A4
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
OCTAL BUF.
d7 d6 d5 d4 d3 d2 d1 d0
OUTPUT
RED1
RED2
RED3
RED4
RED5
RED6
RED7
RED8
NOT
74244
OCTAL BUF.
GREEN1
GREEN2
GREEN3
GREEN4
GREEN5
GREEN6
GREEN7
GREEN8

# APPENDIX B

The code for LEDBOT is provided below.

```
/*
Title           Mine4.c
Programmer      Dhaval Patel
Date            April 23, 2001
Version  1.3
References      A. Antonio Arroy
                        Case histories
                Kevin Harrelson
                        Fuzzy logic theory
                Scott Nortman
                        CPLD assistance and lab help
                Rand Chandler
                        Programming and lab help

Description
A modified program of previous work, combined with design implementations of mine.
The program begins with constants with the decriptions given below.
        Equalizer is used because the left IR and right IR are not perfectly matched.
                It is used for a given threshold to make sure both are relatively similar
                at any given distance.
        Lag left and lag right is used because each motor turns independently,
                therefore dissimiliarities are introduced.  I have determined that the robot
                will fade to the right over some distance, approximately five or more feet.
                I left the lag factors equal because there is not a big enough a difference
                to warent manipulation for my project.
        Mode initialization, mode roaming, mode motion detected, and mode low battery are used
                to correctly reflect the mode the robot is in through the LED display.
        Motion chase is the speed at which the motors will respond to a movement.
        Motion response is the speed at which the motors will turn to a movement.
        Motor left and motor right are difined here according to programming requirements.
        Speed max and speed zero are defined below.
        Tolerance battery is used so the battery check routine will make sure that the
                batteries do not fall below the defined value.
        Tolerance ledge EOW is used to make sure the end-of-world sensor does not fall below
                the defined tolerance in relation to the set value during initialization.
        Tolerance wedge EOW is used to make sure the end-of-world sensor does not rise above
                the defined tolerance in relation to the set value during initialization.
        Tolerance IR is used to equate the differences between the left and right IR.  A
                value is set below since apperantly the left IR does not drop as much as the right.
        Tolerance PIR is used for the motion detectors.  It usually is either true or false,
                but occasionally a one bleeds through as nothing.
        Very near and nothing are the extreme limits for distance measurements which are
                used to set the mid points in the calibration phase.
        Hard left, left, straight, right, and hard right are the motor commands used to
                traverse an area.  The values are used in conjunction with the obstacle avoidance
                matrix.
        Bumper, battery, IR right, IR left, PIR left, EOW, and PIR right are analog
                readings of the various senors on the robot.

The following code was used to test various parts of the robot.
        //This was used to determine the lag found in a motor.
        //The lag compensation allows for straighter traversing.
        while(1){
                motorp(MOTORLEFT, SPEEDMAX / LAGLEFT);
                motorp(MOTORRIGHT, SPEEDMAX / LAGRIGHT);
        }

        //This tests the LED sensor.
        while(1){
                PORTD = 0xFF;
                PORTD = 0x00;
```

```
		wait(500);
		PORTD = 0xFB;
		wait(500);

		printf("\nAll 1s");
		SET_BIT(PORTD, 0x01);
		SET_BIT(PORTD, 0x02);
		SET_BIT(PORTD, 0x04);
		wait(500);

		printf("\n110");
		CLEAR_BIT(PORTD, 0x04);
		wait(500);

		SET_BIT(PORTD, 0x01);
		SET_BIT(PORTD, 0x02);
		SET_BIT(PORTD, 0x04);
		wait(500);

		printf("\n100");
		CLEAR_BIT(PORTD, 0x02);
		CLEAR_BIT(PORTD, 0x04);
		wait(500);

		SET_BIT(PORTD, 0x01);
		SET_BIT(PORTD, 0x02);
		SET_BIT(PORTD, 0x04);
		wait(500);

		printf("\n010");
		CLEAR_BIT(PORTD, 0x01);
		CLEAR_BIT(PORTD, 0x04);
		wait(500);

		SET_BIT(PORTD, 0x01);
		SET_BIT(PORTD, 0x02);
		SET_BIT(PORTD, 0x04);
		wait(500);

		printf("\n000");
		CLEAR_BIT(PORTD, 0x01);
		CLEAR_BIT(PORTD, 0x02);
		CLEAR_BIT(PORTD, 0x04);
		wait(500);
}

//This tests the firing motor.
while(1){
		printf("\nTryin set bit and clear bit");
		SET_BIT(PORTA, 0x10);
		wait(2000);
		CLEAR_BIT(PORTA, 0x10);

		printf("\nTryin motor4");
		motorp(4, 100);
		wait(2000);

		printf("\nTryin motor3");
		motorp(3, 100);
		wait(2000);

		printf("\nTryin motor2");
		motorp(2, 100);
		wait(2000);
}
```

```
        //Testing for turning right and left.
        while(1){
                motorp(MOTORLEFT, 100);
                motorp(MOTORRIGHT, 50);

                wait(500);

                motorp(MOTORLEFT, 3);
                motorp(MOTORRIGHT, 100);

                wait(500);
        }
*/

//Includes
#include <analog.h>
#include <motortjp.h>
#include <clocktjp.h>
#include <isrdecl.h>
#include <vectors.h>
#include <hc11.h>
//#include <stdio.h>

//Constants
#define EQUALIZERIR                     10
#define LAGLEFT                         1
#define LAGRIGHT                        1
#define MODEINITIALIZATION              1
#define MODEROAMING                     2
#define MODEMOTIONDETECTED              3
#define MODELOWBATTERY                  4
#define MOTORLEFT                       0
#define MOTORRIGHT                      1
#define SPEEDMAX                        100
#define SPEEDZERO                       0
#define TOLERANCEBATTERY                120
#define TOLERANCELEDGEEOW               5
#define TOLERANCEWEDGEEOW               10
#define TOLERANCEIR                     115
#define TOLERANCEPIR                    125
#define TURNLEFT                        3
#define TURNRIGHT                       50
#define VNEAR                           127
#define NOTHING                         88
#define HARDLEFT                        -100
#define LEFT                            -TURNLEFT
#define STRAIGHT                        0
#define RIGHT                           TURNRIGHT
#define HARDRIGHT                       100
#define BUMPER                          analog(0)
#define BATTERY                         analog(1)
#define IRRIGHT                         analog(2)
#define IRLEFT                          analog(3)
#define PIRLEFT                         analog(4)
#define EOW                             analog(5)
#define PIRRIGHT                        analog(6)

//Prototypes
void avoidLedges(void);
void batteryCheck(void);
void bumpDetect(void);
void calibrate(void);
void capU(void);
void defuzzify(void);
int  fuzAnd(int, int);
int  fuzNot(int);
int  fuzOr(int, int);
void fuzzify(int);
void init(void);
void ledDisplay(int);
void lookAround(void);
void motionDetect(void);
void move(int);
void obstacleAvoid(void);
void turn(int);

//Globals
int fuzzyHolder[5];
int irFuzzyLeft[5];
int irFuzzyRight[5];
int obstacleAvoidMatrix[25];
int percentageMatrix[25];
int batteryLow, eow, irNear, irFar, irVFar;
int velocity, previousVelocityLeft, previousVelocityRight;
int pirCounter, pirLeftAverage, pirRightAverage;
int fireLeft, fireRight;

void main(void){
	//Initialize all relavent data and wait for back bumper to start.
	init();

	calibrate();

	while(batteryLow == 0){
		capU();
		//printf("\nExited capU, entering avoidLedges");
		avoidLedges();
		//printf("\nExited avoidLegdes, entering bumpDetect");
		bumpDetect();
		//printf("\nExited bumpDetect, entering obstacleAvoid");
		obstacleAvoid();
		//printf("\nExited obstacleAvoid, entering motionDetect");
		motionDetect();
		//printf("\nExited motionDetect, entering batteryCheck");
		batteryCheck();
		//printf("\nExited batterCheck with batteryLow = %d", batteryLow);
	}
}

void avoidLedges(void){
	int sum, a;

	sum = 0;

	for(a = 0; a < 20; a++)
			sum = sum + EOW;

	if((sum / 20) <= (eow - TOLERANCELEDGEEOW) || (sum / 20) >= (eow + 
TOLERANCEWEDGEEOW)){
		/*
		printf("\n\nBackin up because EOW is %d", sum / 20);
		printf("\nwhile eow remains %d\n", eow);
		*/

		motorp(MOTORLEFT, -SPEEDMAX / (LAGLEFT));
		motorp(MOTORRIGHT, -SPEEDMAX / (LAGRIGHT));

		wait(600);

		turn(0);
	}
}
```

```
void batteryCheck(void){
	//printf("\nReading BATTERY as %d", BATTERY);

	//If the battery is low, spin in a counterclockwise circle and flag low battery.
	if(BATTERY <= TOLERANCEBATTERY){
		motorp(MOTORLEFT, -SPEEDMAX / (LAGLEFT));
		motorp(MOTORRIGHT, SPEEDMAX / (LAGRIGHT));

		wait(1200);

		motorp(MOTORLEFT, SPEEDZERO);
		motorp(MOTORRIGHT, SPEEDZERO);

		batteryLow = 1;

		//Turn on the low battery mode for the LED display.
		ledDisplay(MODELOWBATTERY);
	}
}

void bumpDetect(void){
	if((BUMPER > 10) && (BUMPER < 120)){
		motorp(MOTORLEFT, -SPEEDMAX / (LAGLEFT));
		motorp(MOTORRIGHT, -SPEEDMAX / (LAGRIGHT));

		wait(600);

		turn(0);
	}
}

void calibrate(void){
	//By placing the robot close to a wall, approximately where
	//it should turn to avoid an obstacle, you can extrapolate
	//NEAR, FAR, and VFAR, with NOTHING and VNEAR constant
	int calibrate, sum, sumLeft, sumRight, a;

	calibrate = 0;

	while(calibrate == 0){
		calibrate = 1;

		sum = (sumLeft = (sumRight = 0));

		wait(60);

		for(a = 0; a < 20; a++){
			sum = sum + EOW;
			sumLeft = sumLeft + IRLEFT;
			sumRight = sumRight + IRRIGHT;
		}

		eow = sum / 20;
		irNear = ((sumLeft / 20) + (sumRight / 20)) / 2;
		irVFar = NOTHING + ((irNear - NOTHING) / 3);
		irFar = NOTHING + (2 * (irNear - NOTHING) / 3);

		if(irNear <= 0 || irVFar <= 0 || irFar <= 0 || irNear <= irFar || irVFar == NOTHING){
			calibrate = 0;

			motorp(MOTORLEFT, SPEEDMAX / (2 * LAGLEFT));
			motorp(MOTORRIGHT, SPEEDMAX / (2 * LAGRIGHT));

			wait(50);
```

```
				motorp(MOTORLEFT, SPEEDZERO);
				motorp(MOTORRIGHT, SPEEDZERO);
			}

			if(VNEAR <= irNear){
				calibrate = 0;

				motorp(MOTORLEFT, -SPEEDMAX / (2 * LAGLEFT));
				motorp(MOTORRIGHT, -SPEEDMAX / (2 * LAGRIGHT));

				wait(50);

				motorp(MOTORLEFT, SPEEDZERO);
				motorp(MOTORRIGHT, SPEEDZERO);
			}

			/*
			Debugging check
			printf("\n Battery   = %d", BATTERY);
			printf("\n EOW       = %d", eow);
			printf("\n Very Near = %d", VNEAR);
			printf("\n Near      = %d", irNear);
			printf("\n Far       = %d", irFar);
			printf("\n Very Far  = %d", irVFar);
			printf("\n Nothing   = %d", NOTHING);
			*/
		}
}

void capU(void){
		//Fire a cap only if there is an object was detected from both PIR detectors.
		if((fireLeft == 1) && (fireRight == 1)){
			SET_BIT(PORTA, 0x10);
			wait(2000);
			CLEAR_BIT(PORTA, 0x10);

			fireLeft = (fireRight = 0);
		}
}

void defuzzify(void){
		int hardLeft, left, straight, right, hardRight, a, b, c, d;

		hardLeft = (left = (straight = (right = (hardRight = 0))));

		//printf("\n");

		//And the inputs
		for(a = 0; a < 5; a++){
			for(b = 0; b < 5; b++){
				percentageMatrix[(a * 5) + b] = fuzAnd(irFuzzyLeft[a], irFuzzyRight[b]);

				/*
			printf("%d\t", percentageMatrix[(a * 5) + b]);

			if(b == 4)
				printf("\n");
			*/
			}
		}

		/*
		printf("\n");

		for(c = 0; c < 25; c++){
			printf("%d\t", percentageMatrix[c]);
```

```
        if(c == 4 || c == 9 || c == 14 || c == 19)
                printf("\n");
}
*/

//Or the result
for(d = 0; d < 25; d++){
        if(obstacleAvoidMatrix[d] == HARDLEFT){
                /*
                printf("\nHARDLEFT");
                printf("\n%d", fuzOr(percentageMatrix[d], hardLeft));
                */

                hardLeft = fuzOr(percentageMatrix[d], hardLeft);
        }
        if(obstacleAvoidMatrix[d] == LEFT){
                /*
                printf("\nLEFT");
                printf("\n%d", fuzOr(percentageMatrix[d], left));
                */

                left = fuzOr(percentageMatrix[d], left);
        }
        if(obstacleAvoidMatrix[d] == STRAIGHT){
                /*
                printf("\nSTRAIGHT");
                printf("\n%d", fuzOr(percentageMatrix[d], straight));
                */

                straight = fuzOr(percentageMatrix[d], straight);
        }
        if(obstacleAvoidMatrix[d] == RIGHT){
                /*
                printf("\nRIGHT");
                printf("\n%d", fuzOr(percentageMatrix[d], right));
                */

                right = fuzOr(percentageMatrix[d], right);
        }
        if(obstacleAvoidMatrix[d] == HARDRIGHT){
                /*
                printf("\nHARDRIGHT");
                printf("\n%d", fuzOr(percentageMatrix[d], hardRight));
                */

                hardRight = fuzOr(percentageMatrix[d], hardRight);
        }
}

/*
printf("\n%d", fuzOr(33, 24));
printf("\n%d", fuzOr(0, 2));
printf("\n%d", fuzOr(2, 0));
printf("\nhardLeft  = %d", hardLeft);
printf("\nleft      = %d", left);
printf("\nstraight  = %d", straight);
printf("\nright     = %d", right);
printf("\nhardRight = %d", hardRight);
*/

//Compute the weighted average output
velocity = ((hardLeft * HARDLEFT) +
                        (left * LEFT) +
                        (straight * STRAIGHT) +
                        (right * RIGHT) +
                                        (hardRight * HARDRIGHT))
                                        / (hardLeft + left + straight + right + hardRight);
}

int fuzAnd(a, b){
        if(a < b)
                return a;
        else
                return b;
}

int fuzNot(a){
        return (255-a);
}

int fuzOr(a, b){
        if(a > b)
                return a;
        else
                return b;
}

void fuzzify(a){
        int current = VNEAR;
        int percentage1, percentage2, spanGreater, spanLesser, b, c;

        percentage1 = (percentage2 = (spanGreater = (spanLesser = 0)));

        /*
        printf("\np1 = %d", percentage1);
        printf("\np2 = %d", percentage2);
        printf("\nGreater = %d", spanGreater);
        printf("\nLesser  = %d", spanLesser);
        */

        //Take numbers larger than or less than the limits and make them the limit
        //If exactly on the defined value then make spans the same
        if(a >= VNEAR)
                spanGreater = (spanLesser = VNEAR);
        if(a == irNear)
                spanGreater = (spanLesser = irNear);
        if(a == irFar)
                spanGreater = (spanLesser = irFar);
        if(a == irVFar)
                spanGreater = (spanLesser = irVFar);
        if(a <= NOTHING)
                spanGreater = (spanLesser = NOTHING);

        while(spanLesser == 0){
                //printf("\ncurrent = %d", current);

                if(a < current)
                        spanGreater = current;
                else
                        spanLesser = current;

                if(current == irVFar)
                        current = NOTHING;
                if(current == irFar)
                        current = irVFar;
                if(current == irNear)
                        current = irFar;
                if(current == VNEAR)
                        current = irNear;
        }
```

```
	/*
	printf("\na = %d", a);
	printf("\nspanGreater = %d", spanGreater);
	printf("\nspanLesser  = %d", spanLesser);
	*/

	//Take the differnce of the spans and divide by total span
	if(spanGreater == spanLesser)
		percentage1 = (percentage2 = 100);
	else{
		percentage1 = ((100 * (a - spanGreater)) / (spanGreater - spanLesser));
		percentage2 = ((100 * (a - spanLesser)) / (spanGreater - spanLesser));
	}

	//Take absolute value
	if(percentage1 < 0)
		percentage1 = -percentage1;
	if(percentage2 < 0)
		percentage2 = -percentage2;

	/*
	printf("\np1 = %d", percentage1);
	printf("\np2 = %d", percentage2);
	*/

	//First percentage goes to spanLesser spot
	//Second percentage goes to spanGreater
	//The fuzzy left and right ir arrays are of the following format:
	//[VN, N, F, VF, NOT]
	for(b = 0, current = VNEAR; b < 5; b++){
		/*
		printf("\ncurrent = %d", current);
		printf("\nspanGreater = %d", spanGreater);
		printf("\nspanLesser  = %d", spanLesser);
		*/

		if(spanLesser == current)
			fuzzyHolder[b] = percentage1;

		if(spanGreater == current)
			fuzzyHolder[b] = percentage2;

		if(current == irVFar)
			current = NOTHING;
		if(current == irFar)
			current = irVFar;
		if(current == irNear)
			current = irFar;
		if(current == VNEAR)
			current = irNear;
	}

	/*
	//Debuggin check
	printf("\n");
	for(c = 0; c < 5; c++)
		printf("%d ", fuzzyHolder[c]);
	*/
}

void init(void){
	int a;

	//Make sure globals are set to zero initially
	batteryLow = (eow = (irNear = (irFar = (irVFar = 0))));
	velocity = (previousVelocityLeft = (previousVelocityRight = SPEEDZERO));
	pirCounter = (pirLeftAverage = (pirRightAverage = 0));
	fireLeft = (fireRight = 0);

	init_analog();
	init_motortjp();
	init_clocktjp();

	//Turn the LED display on for initializaion mode.
	//First write all ones to the address to make the port output.
	//Then set to initialization mode.
	DDRD = 0xFF;
	wait(250);
	ledDisplay(MODEINITIALIZATION);

	//Make the obstacle avoidance matrix
	for(a = 0; a < 25; a++){
		if(a <= 4 || a == 7)
			obstacleAvoidMatrix[a] = HARDRIGHT;
		if(a == 5 || a == 10 || a == 11 || a == 15 || a == 20)
			obstacleAvoidMatrix[a] = HARDLEFT;
		if(a == 6 || a == 16 || a == 21)
			obstacleAvoidMatrix[a] = LEFT;
		if(a == 8 || a == 9)
			obstacleAvoidMatrix[a] = RIGHT;
		if((a > 11 && a < 15) || (a > 16 && a < 20) || (a > 21))
			obstacleAvoidMatrix[a] = STRAIGHT;

		/*
		printf("%d\t", obstacleAvoidMatrix[a]);

		if(a == 4 || a == 9 || a == 14 || a == 19)
			printf("\n");
		*/
	}

	//Turn on IR
	*(unsigned char *) 0x7000 = 0x07;

	//Press the rear bumper to start the program
	while(BUMPER < 120);
}

void ledDisplay(mode){
	//Bits 0, 1, and 2 are used on Port D to act as switches for the LED display.
	//Bit 0 controls the color, zero is green and one is red.
	//Bit 1 controls the tpye of display, zero is running and one is flashing.
	//Bit 2 enables the logic, zero is on and one is off.
	//First set bits 0, 1, 2 to ones to clear and disable.
	SET_BIT(PORTD, 0x01);
	SET_BIT(PORTD, 0x02);
	SET_BIT(PORTD, 0x04);

	//This will set to 011, then 010.
	if(mode == MODEINITIALIZATION){
		//printf("\nInitialization");

		CLEAR_BIT(PORTD, 0x01);
	}

	//This will set to 001, then 000.
	if(mode == MODEROAMING){
		//printf("\nRoaming");

		CLEAR_BIT(PORTD, 0x01);
		CLEAR_BIT(PORTD, 0x02);
	}
	/*
	//Note that motion detected mode does not require any bit manipulation.
	if(mode == MODEMOTIONDETECTED){
		printf("\nMotion Detected");
	}
	*/

	//This will set to 101, then 100.
	if(mode == MODELOWBATTERY){
		//printf("\nLow Battery");

		CLEAR_BIT(PORTD, 0x02);
	}

	//Now enable after waiting.
	wait(250);
	CLEAR_BIT(PORTD, 0x04);
}

void lookAround(void){
	//This routine makes the robot appear to look around
	//in response to seeing something move.
	//The motor movements should result in the original orientation.
	motorp(MOTORLEFT, SPEEDZERO);
	motorp(MOTORRIGHT, SPEEDZERO);

	wait(250);

	motorp(MOTORLEFT, -SPEEDMAX / (LAGLEFT));
	motorp(MOTORRIGHT, SPEEDMAX / (LAGRIGHT));

	wait(250);

	motorp(MOTORLEFT, SPEEDMAX / (LAGLEFT));
	motorp(MOTORRIGHT, -SPEEDMAX / (LAGRIGHT));

	wait(500);

	motorp(MOTORLEFT, -SPEEDMAX / (LAGLEFT));
	motorp(MOTORRIGHT, SPEEDMAX / (LAGRIGHT));

	wait(250);
}

void motionDetect(void){
	int motion, speedLeft, speedRight;

	speedLeft = previousVelocityLeft;
	speedRight = previousVelocityRight;

	motion = PIRLEFT - PIRRIGHT;

	/*
	printf("\n\nMotion is      %d", motion);
	printf("\nSmooth PIR Left  %d", pirLeftAverage);
	printf("\tWhile PIRLEFT    %d", PIRLEFT);
	printf("\nSmooth PIR Right %d", pirRightAverage);
	printf("\tWhile PIRRIGHT   %d\n", PIRRIGHT);
	*/

	if(motion != 0){
		//Turn the LED display on for motion mode.
		ledDisplay(MODEMOTIONDETECTED);

		motorp(MOTORLEFT, SPEEDZERO);
		motorp(MOTORRIGHT, SPEEDZERO);

		wait(250);
	}

	//Sence the PIR sensors are either on or off, moving to a motion will take guess work
	if(motion >= TOLERANCEPIR){
		//printf("\nDetected Left Motion");

		turn(LEFT);

		//Flag for possible firing.
		fireLeft = 1;
	}

	//Detect something on the right, then move toward it.
	if(-motion >= TOLERANCEPIR){
		//printf("\nDetected Right Motion");

		turn(RIGHT);

		//Flag for possible firing.
		fireRight = 1;
	}
}

void move(pimpin){
	int speedLeft, speedRight;

	speedLeft = (speedRight = SPEEDMAX);

	if(pimpin != 0){
		if(pimpin < 0)
			speedLeft = pimpin;
		else
			speedRight = pimpin;
	}

	//Smoothing attempt for turns
	speedLeft = ((previousVelocityLeft) + (9 * speedLeft)) / 10;
	speedRight = ((previousVelocityRight) + (9 * speedRight)) / 10;

	previousVelocityLeft = speedLeft;
	previousVelocityRight = speedRight;

	motorp(MOTORLEFT, speedLeft / (LAGLEFT));
	motorp(MOTORRIGHT, speedRight / (LAGRIGHT));
}

void obstacleAvoid(void){
	int sumLeft, sumRight, a, b, c;

	sumLeft = (sumRight = 0);

	//Turn the LED display on for roam mode.
	ledDisplay(MODEROAMING);

	//printf("\nL\tR\n");

	//Smoothing function for IR readings
	for(a = 0; a < 20; a++){
		/*
		printf("%d\t", IRLEFT);
		printf("%d\n", IRRIGHT);
		*/
```

```
			sumLeft = sumLeft + IRLEFT;
			sumRight = sumRight + IRRIGHT;
		}

		/*
		printf("\nsumLeft is %d", sumLeft);
		printf("\nFuzzifing left %d", sumLeft / 20);
		*/

		//Fuzzify the IR readings and place them into appropriate global array
		//Must have a IR equalizer for the left eye
		if(sumLeft/20 < TOLERANCEIR)
			fuzzify((sumLeft / 20) - EQUALIZERIR);
		else
			fuzzify(sumLeft / 20);

		//printf("\nL: ");

		for(b = 0; b < 5; b++){
			irFuzzyLeft[b] = fuzzyHolder[b];
			fuzzyHolder[b] = 0;

			//printf("%d ", irFuzzyLeft[b]);
		}

		/*
		printf("\nsumRight is %d", sumRight);
		printf("\nFuzzifing right %d", sumRight / 20);
		*/

		fuzzify(sumRight / 20);

		//printf("\nR: ");

		for(c = 0; c < 5; c++){
			irFuzzyRight[c] = fuzzyHolder[c];
			fuzzyHolder[c] = 0;

			//printf("%d ", irFuzzyRight[c]);
		}

		defuzzify();

		//printf("\nvelocity = %d\n", velocity);

		move(velocity);
	}

	void turn(where){
		int i;
		unsigned rand;

		//printf("\nwhere is %d", where);

		if(where == RIGHT){
			//printf("\nTurned Right");

			motorp(MOTORLEFT, SPEEDMAX);
			motorp(MOTORRIGHT, TURNRIGHT);

			wait(500);

			move(STRAIGHT);
		}

		if(where == LEFT){
			//printf("\nTurned Left");

			motorp(MOTORLEFT, TURNLEFT);
			motorp(MOTORRIGHT, SPEEDMAX);

			wait(500);

			move(STRAIGHT);
		}

		if(where == 0){
			rand = TCNT;

			if(rand & 0x0001){
				//turn left
				motorp(MOTORLEFT, -SPEEDMAX / (LAGLEFT));
				motorp(MOTORRIGHT, SPEEDMAX / (LAGRIGHT));
			}
			else{
				//turn right
				motorp(MOTORLEFT, SPEEDMAX / (LAGLEFT));
				motorp(MOTORRIGHT, -SPEEDMAX / (LAGRIGHT));
			}

			i=(rand % 1024);

			if(i > 250)
				wait(i);
			else
				wait(250);
		}
	}
```