5-2009

Intelligent-navigating, Maze-mapping, and Maze-solving Robot Equipped with Sensors, Compass, and GPS System

Yunan Yuan

Intelligent-navigating, Maze-mapping, and Maze-solving Robot Equipped with Sensors, Compass, and GPS System

by

Yunan Yuan

Undergraduate honors thesis under the direction of

Dr. Jerry Trahan

Department of Electrical and Computer Engineering

Submitted to the LSU Honors College in partial fulfillment of the Upper Division Honors Program.

May, 2009

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, Louisiana

Intelligent-navigating, Maze-mapping, and Maze-solving Robot Equipped with Sensors, Compass, and GPS System

Yunan Yuan

Honor's thesis at: Louisiana State University

Conducted at: Department of Electrical and Computer Engineering

Supervisor: **Jerry Trahan, Ahmed A. El-Amawy**
Department of Electrical and Computer Engineering

Committee: **Jerry Trahan , Bahadir K. Gunturk**
Department of Electrical and Computer Engineering
Ye-Sho Chen
Department of Information Systems and Decision Sciences,
E.J. Ourso College of Business

ABSTRACT

This paper documents the process of designing, assembling, programming, interfacing, and testing a robot in detail. The three main functionalities of the robot are: intelligent navigation, maze solving, and maze mapping.

This project constitutes an undergraduate senior design project in the field of computer engineering. It involves the calibration, configuration, and interfacing of various components, such as microprocessor, motors, sensors, compass, and GPS system. Moreover, much research is conducted on various maze-solving algorithms. Finally, assembling and testing the robot involve many design decisions. This paper discusses all of the above and contemplates on the value of the project as well as the potential applications for the robot.

DEDICATION

This thesis is dedicated to my parents, **Qingzhong Yuan** and **Jinhui Liu**, who are the source of inspiration and encouragement for everything I have achieved in my college career.

ACKNOWLEDGEMENTS

I would like to thank Professor **Ahmed A. El-Amawy**, whose class is the source of inspiration for this project, for giving me full guidance and support for this project until its completion.

I would like to thank Professor **Jerry Trahan**, who is the supervisor for the second half of the project, for supervising the entire drafting process and making this paper possible.

I would like to thank **LSU Honors College** and **ECE Department** for funding this project.

TABLE OF CONTENTS

I. INTRODUCTION

This project consists of designing, assembling, programming, interfacing, and testing a robot. The three main objectives for the robot are: intelligent navigation, maze solving, and maze mapping.

This project constitutes an undergraduate senior design project in the field of computer engineering. The original plan was conceived in EE4750, Microprocessor Interfacing Techniques. EE4750 is a required class for first-semester seniors majoring in computer engineering. It consists of a lecture that discusses "theory and design techniques of microprocessor interfaces to memory and input/output devices," and a laboratory that provides students with materials and assistance in a relevant project. During Fall 2008, I took the course and worked on constructing an intelligent-navigating robot in a 2-person team under the supervision of Professor El-Amawy. Throughout the semester, we worked on the hardware calibration / configuration and software programming / interfacing of components[1] and assembled the robot, eventually making it intelligent-navigating. That is, the robot can travel from one specified GPS coordinate to another with a given path, detecting and avoiding any obstacles on the way.

After the completion of the course project, I proposed to add two additional functionalities to the robot: maze-solving and maze-mapping[2]. The proposal was reviewed and accepted by my EE4750 professor and then-supervisor Dr. El-Amawy[3]. With funding from the Honors College[4], I conducted research on maze-solving algorithms, added the required hardware and programmed the

[1] The components include two motors, two sensors, a compass, and a GPS system.
[2] Functionalities are explained in detail in the following section.
[3] Professor El-Amawy retired between Fall 2008 and Spring 2009. At the beginning of Spring 2009, Professor Jerry Trahan became the current thesis supervisor.
[4] The Honors College granted me the Tiger Athletic Foundation Research Scholarship.

robot. The robot can successfully complete the following tasks: traverse a perfect maze of any size, map the maze on a LCD screen, and darken all dead ends.

This paper is structured in five sections: Introduction, Objectives, Implementation, Results, and Conclusion, followed by References and Appendix. The Objectives Section describes the detailed deliverables for the robot. The Implementation Section lists individual components and discusses their calibration/configuration, programming, and interfacing in detail; it also explains the maze-solving algorithms and the assembling process. The Results Section uses two videos to demonstrate the robot meeting the objectives. Finally, the Conclusion Section evaluates the robot and contemplates on things learned from this project. The Appendix includes assembly code and relevant datasheets.

II. OBJECTIVES

The robot completes the following tasks:

1. Intelligent-Navigating

A. Navigate between two points on earth given their latitudes and longitudes.

B. Detect and avoid obstacles.

C. Stay on the given path and automatically correct deviations.

2. Maze-Mapping

A. Traverse a perfect maze[5] from S to F using a right-hand rule.

B. Traverse the maze from F to S using a left-hand rule.

C. Display the maze layout on an LCD screen.

3. Maze-Solving

A. Solve the maze and display the solution on the LCD screen by darkening all dead ends.

B. Traverse the maze from S to F using the shortest path and avoid all dead ends.

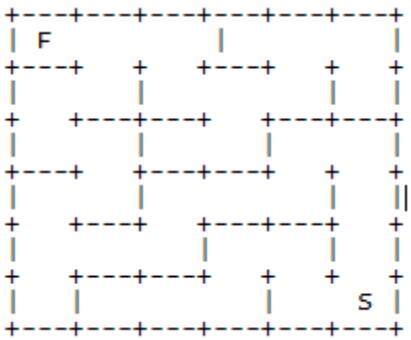


Figure 1 Sample 6x6 Perfect Maze

[5] A perfect maze is defined as a maze that has one and only one path from any point in the maze to any other point. That is, the maze has no inaccessible sections, no circular paths, no open areas.

III. IMPLEMENTATION

This section consists of two parts. Part A lists individual components and discusses how to calibrate, program and interface each one in detail. Part B explains how to assemble and program all components to construct the robot.

A. CONFIGURING COMPONENTS

The main hardware components include: microcontroller, motors, sensors, LCD display, GPS, and compass.

1. Microcontroller

Figure 2 BasicATOM-24 Development Kit

The most important component, the "brain" of the robot, is the microcontroller. This project uses the BasicATOM-24 Development Kit, which consists of BasicATOM Universal Development Board, BaseATOM-M module, accompanying software, manual and DB-9 serial cable.

The programming environment for the BasicATOM microprocessor is the BasicMicro ATOM IDE. The language used is assembly language.

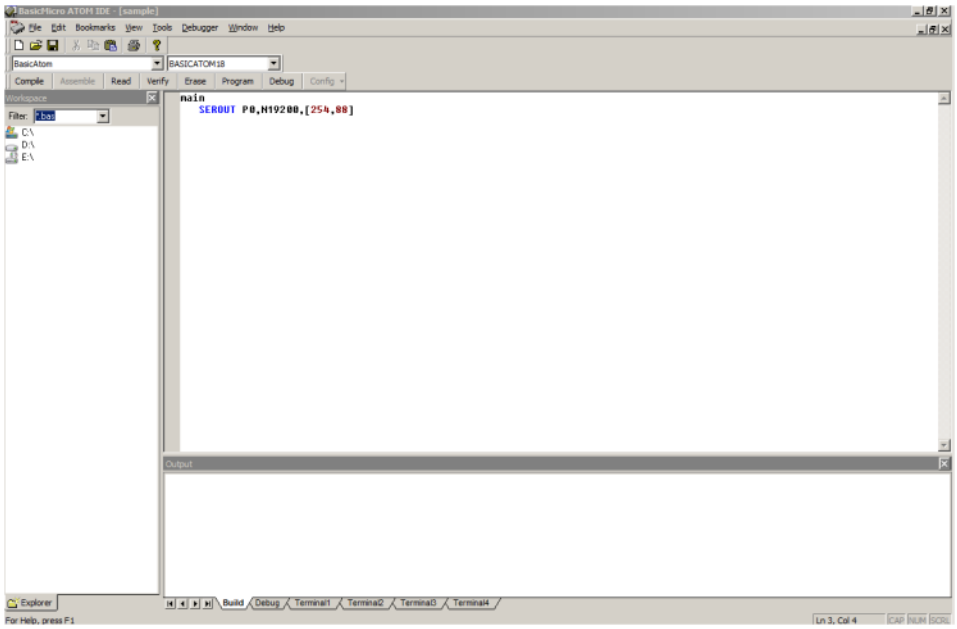


Figure 3 BasicMicro ATOM IDE

The microprocessor is connected to the programming machine with the DB-9 serial cable. Furthermore, the microprocessor is powered with three 9V batteries in parallel.

2. Motors

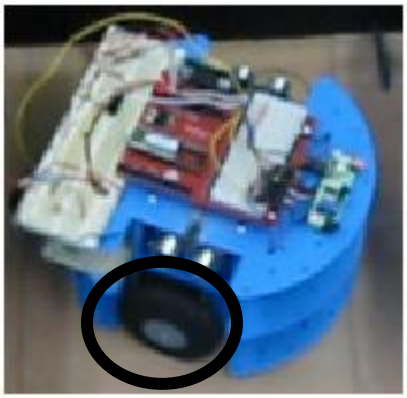


Figure 4 Servo Motor

Two servo motors are mounted on the robot to allow it to move, one on each side.

A Servo is a small device that has an output shaft. This shaft can be positioned to specific angular positions and speed by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft.[6]

The signal value to achieve desired effect, such as moving forward fast and moving backward slow, varies from one motor to another. In order to have the robot move or turn at a desired speed, the two motors are tuned by trial and error during the calibration phase, until the desired effect is obtained by the combination of two signal values.

After the desired signal values are obtained for the two particular servo motors, subroutines are written for various movements. These subroutines can be called in for loops to prolong the motion. Three sample subroutines are shown below

[6] Visit http://www.seattlerobotics.org/guide/servos.html for more information.

```
FORWARD_FAST:
    LOW LeftMotor
    LOW RightMotor
    PULSOUT LeftMotor, 3340
    PULSOUT RightMotor, 10
RETURN

TURN_SLIGHT_RIGHT:
    ;==========
    FOR i=1 to 2
    LOW RightMotor
    LOW LeftMotor
    PULSOUT LeftMotor, 3000
    PULSOUT RightMotor, 1525
    NEXT
    ;PAUSE 100
RETURN

TURN_SLIGHT_LEFT:
    ;==========
    FOR i=1 to 2
    LOW RightMotor
    LOW LeftMotor
    PULSOUT LeftMotor, 1894
    PULSOUT RightMotor, 45
    NEXT
;   PAUSE 500
RETURN
```

3. Sensors

Sensors are used to detect the presence and sometimes the distance of obstacles. The robot is equipped with two types of sensors: infrared sensors and optical sensors.

- Infrared sensors

The two infrared sensors are located in the front of the robot. The infrared sensors are IRPD Ver 3.0 Infrared Proximity Detector produced by Lynxmotion, Inc[7].

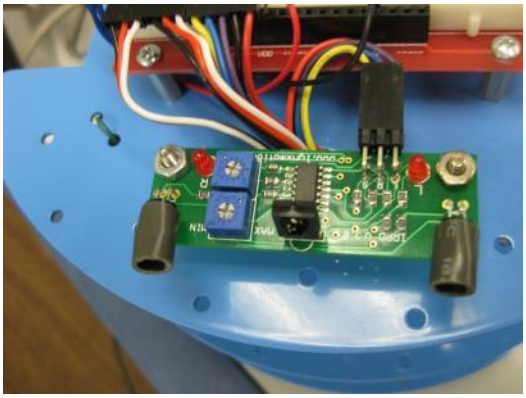


Figure 5 Infrared Sensors

As illustrated below, there are two power connections (Vdc and ground). There are three microprocessor connections, one for the microprocessor to output a pulse signal to the two sensors, and the other two to send the sensors' response back to the microprocessor.

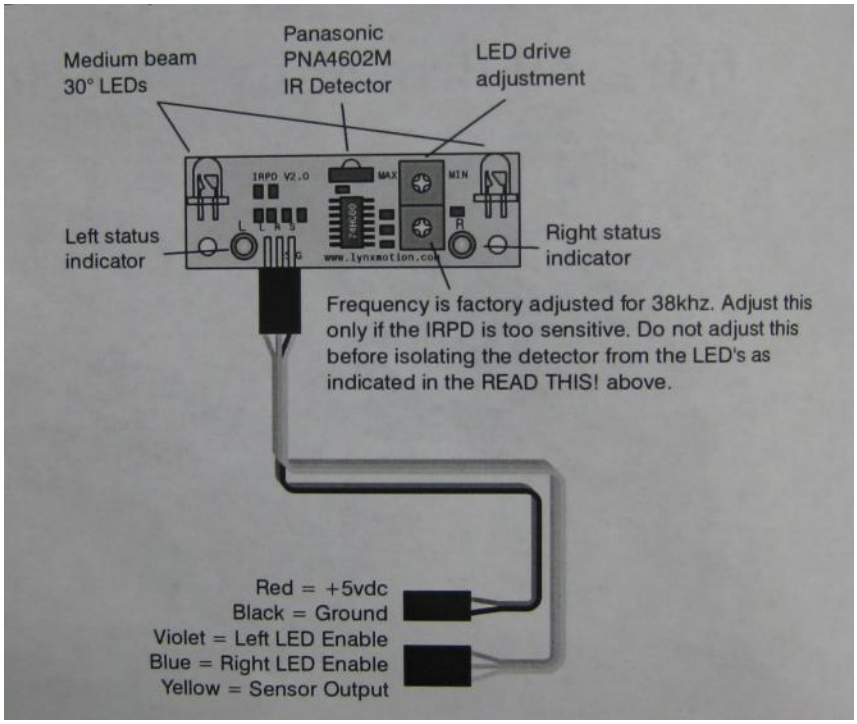


Figure 6 Power Connection

[7] Visit http://www.lynxmotion.com for more details.

The two IR Detectors are very sensitive. To prevent false readings, a short section of black tubing is put on the back and sides of each IR Detector. The PC board is mounted on top of the robot. The optimum height is from 4" to 6" from the ground, which is allowed by the height of this particular robot. As advised by the datasheet, the sensor is mounted horizontally as close to the front of the robot as possible. There are no moving parts of the robot within the field of view.

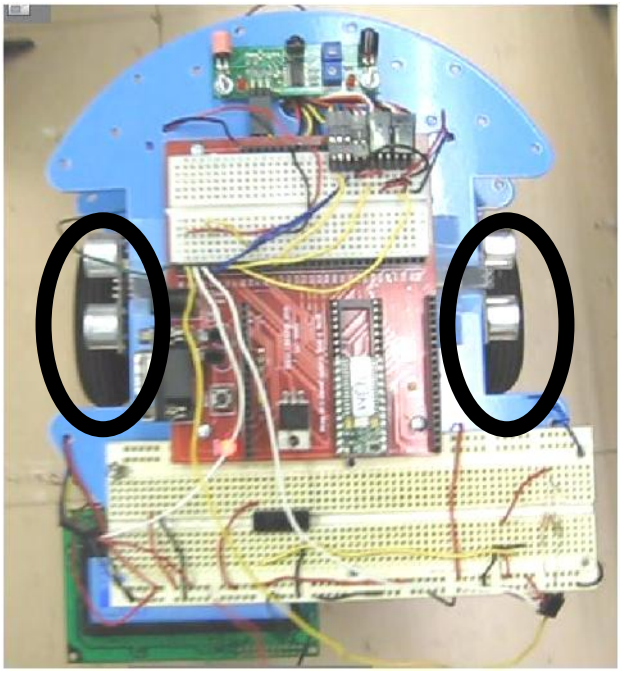
To detect obstacles, a pulse is sent to each sensor, which sends back a signal in response. The signal contains a Boolean value indicating whether an obstacle is detected by the sensor. A sample code is shown:

```
HIGH FrontPingR
PAUSE 10
FrontR = in12
LOW FrontPingR

HIGH FrontPingL
PAUSE 10
FrontL = in12
LOW FrontPingL
```

- **Optical Sensors**

Two pairs of optical sensors are located on both sides of the robot. The optical sensors are PING))) Ultrasonic Sensors produced by Parallax.[8] They detect obstacles AND measure their distance from the robot.



The PING))) sensor measures distance using sonar; an ultrasonic pulse is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo return.

Output from the PING))) sensor is a variable-width pulse that corresponds to the distance to the target.

Figure 7 PING))) Sensors

The PING))) Sensor has only 3 connections: power, ground, and signal. To get the distance from the obstacle to the sensor, the code below is used:

```
LOW RightPing
PAUSE 2
PULSOUT RightPing, Trigger
PULSIN  RightPing, IsHigh, RightRawDist
```

[8] Visit http://www.parallax.com/ for more details.

4. LCD Display

An LCD display is used to draw the maze traversed by the robot and darken the dead ends. The LCD display used is MATRIX ORBITAL GLK12232-25-wbl.

The GLK12232-25-WBL comes equipped with the following features[9]:

- 122 x 32 pixel graphics display,
- Text display using built-in or user-supplied fonts,
- Adjustable contrast,
- Backlighting, and
- Keypad interface RS-232 or I2C communications.

To set up the LCD display, three connections are sufficient: to power, to ground, and to the microprocessor. No calibration is required. Reset may be occasionally needed.

Figure 8 Setting up LCD Display

The display provides a simple command structure to allow both text and graphics to be transferred to the screen, for example:

```
SEROUT LCD,N19200,[254,120,1,X1,Y1,X2,Y2]
```

[9] For more details, visit www.bipom.com/lcds/manuals/**glk12232-25-wbl**.pdf

5. Compass

A compass is used to detect deviations from a given direction. In this project, the compass is Devantech Magnetic Compass Module CMPS03. This compass module is designed for use in robots as an aid to navigation. "The aim was to produce a unique number to represent the direction the robot is facing."[10]

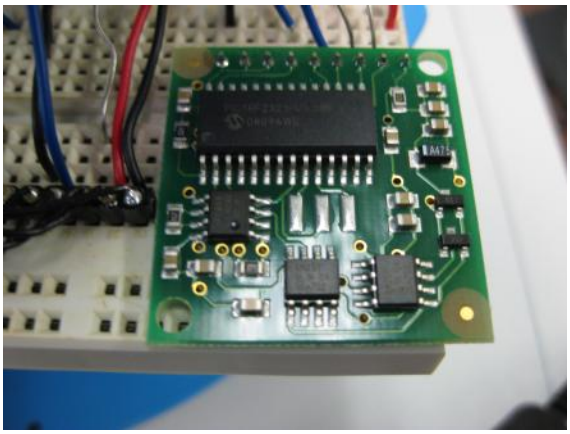


Figure 9 Compass

The compass uses a magnetic field sensor, which is sensitive enough to detect the Earth's magnetic field. The output from two of them mounted at right angles to each other is used to compute the direction of the horizontal component of the Earth's magnetic field.

The compass module can be easily connected to the microprocessor, as shown below

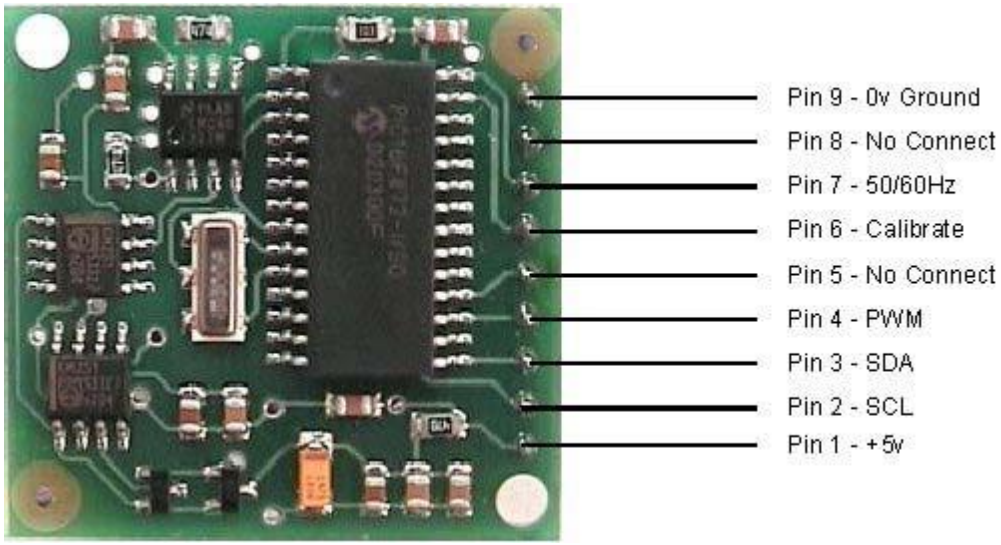


Figure 10 Compass Connections

[10] For more information, visit http://www.robotstorehk.com/CMPS03_release.pdf

Before using the compass, it must be calibrated. Pin 6 is used to calibrate the compass. To calibrate the compass, the calibrate pin is set low and then high again for each of the four major compass points: North, East, South, and West.[11]

The following sample code is used to get readings from the compass:

```
PULSIN COMPASS,1,signal
```

[11] A simple push switch wired from pin6 to 0v is OK for this. For details, see datasheet.

6. GPS

A GPS system is used to measure the real-time longitude and latitude coordinates of the robot in intelligent-navigation. The GPS used is the EM-406A GPS module developed by GlobalSat Corp.[12] The GPS module is based on the SiRF Star III chipset. It includes on-board voltage regulation, LED status indicator, battery backed RAM, and a built-in patch antenna.

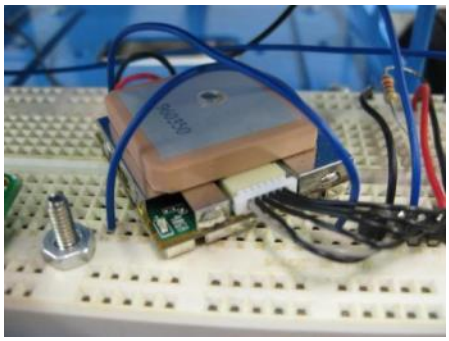
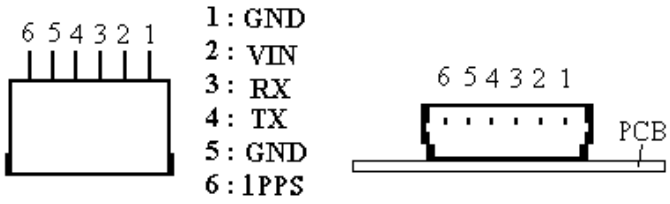


Figure 11 GPS

As shown below, the GPS has the following connections: VIN (power), two GNDs (grounds), and RX (the main transmission channel for outputting navigation and measurement data). In particular, a voltage-divider circuit needs to be built to supply the correct amount of VIN. Also, a buffer is used between RX and the microprocessor pin. The GPS needs approximately 10 minutes to activate itself after it is powered.



The following command reads the output string from the GPS in the NMEA 0183 data protocol[13]:

```
SERIN GPS,N4800,[WAIT("$GPGGA"),str COORDINATES\47]
```

[12] Visit **www.globalsat.com.tw** for more information.

[13] NMEA 0183 (or NMEA for short) is a combined electrical and data specification for communication between marine electronic devices such as echo sounder, sonars, anemometer (wind speed and direction), gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the U.S.-based National Marine Electronics Association.

B. IMPLEMENTING PROPOSED FUNCTIONALITIES

1. For Intelligent-Navigation

Hardware

For intelligent-navigation, the robot is expected to meet three requirements. Different components from the previous section are chosen for each requirement.

First, the robot can follow a given path and travel to a given GPS coordinate. The given path comprises connected straight lines; their end points are measured GPS coordinates, and at each joining point, the two lines form a right angle. An example of a path is given:

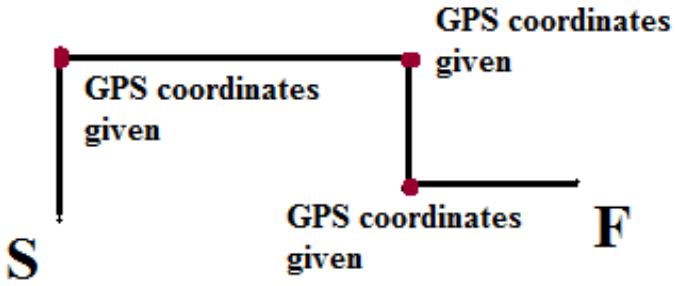
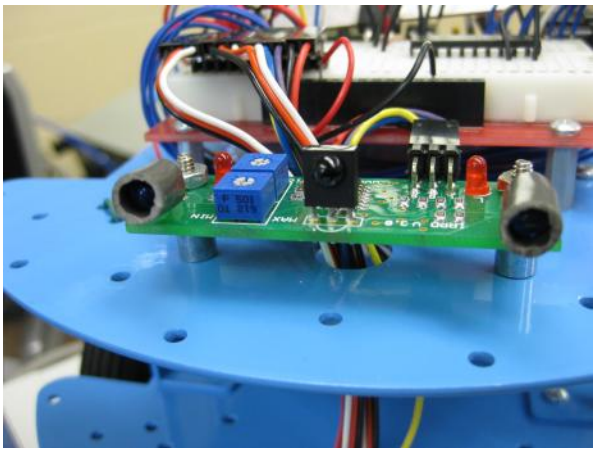


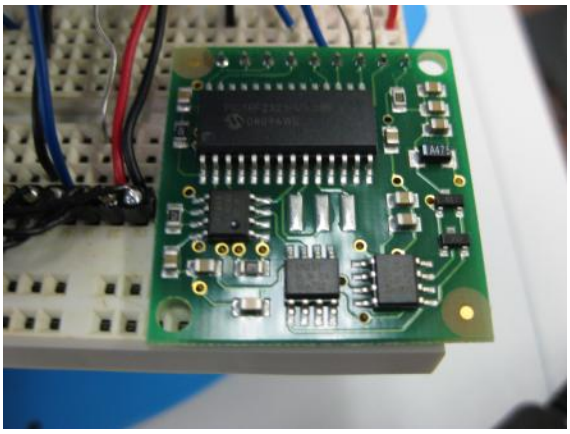
Figure 12 Sample path

For this purpose, the two motors are mounted on two sides of the robot to allow movements. The GPS system is mounted on top of the robot to monitor the robot's GPS location.

Second, the robot can detect and avoid any obstacles on the path. For this purpose, sensors are needed. Since the path consists of only straight lines, the obstacles on the path can only appear right in front of the robot. Thus, only the infrared sensors, which are mounted to the front of the robot, are needed.



The third requirement of intelligent-navigation is that the robot can stay on the given path, that is, the robot can detect deviation and make adjustment to return to the path. For this purpose, the compass is used to monitor the robot's direction.



Software:

An algorithm is designed to achieve the three requirements. To avoid running into obstacles, obstacle monitoring should occur as often as possible. On the other hand, GPS monitoring should occur only frequently enough so that the robot will not deviate too much from the given path. Similarly, deviation monitoring should occur only frequently enough so that the deviation is neither too small to observe nor too big to correct.

The algorithm proceeds as follow: The robot moves forward in small bursts. Inside each burst, the robot constantly nudges forward and checks if there is an obstacle ahead. Between each burst, the robot checks if it has deviated from its path and if has reached the specified GPS coordinate. The pseudo-code is given below

```
should_turn = false

WHILE(should_turn == false)

    ;///one bust
    FOR counter = 0 to burst
        GOSUB NUDGE

        ;///obstacle monitoring
        GOSUB READ_SENSOR
        IF(obstacle_detected)
            GOSUB AVOID
        ENDIF
    END

    ;///deviation monitoring
    GOSUB READ_COMPASS
    IF(deviation)
        GOSUB ADJUST
    ENDIF

    ;///GPS monitoring
    GOSUB READ_GPS
    GOSUB CHECK_IF_SHOULD_TURN

END
```

2. For Maze-solving and Maze-mapping

Hardware: For maze-solving and maze-mapping, the robot is expected to meet two requirements. Different components from the previous section are chosen for each requirement.

First, the robot is expected to traverse a perfect maze using a right-hand rule or left-hand rule. Obviously two motors are needed. In addition, the robot needs to determine whether there are obstacles to its front, left and right. Thus, two optical sensors are mounted on both sides of the robot.

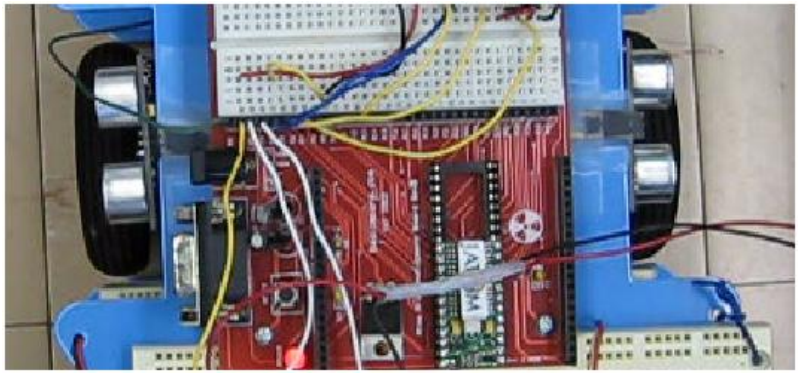


Figure 13 Optical Sensors

Second, the robot is expected to display the traversed maze and then "solve" it by darkening its dead ends. For this purpose, an LCD display is mounted on the robot. In addition, in order to see both the maze and the "solved" maze, a switch is connected to the LCD display. When the button is clicked once, the maze is drawn; when the button is clicked once again, the dead ends are darkened, leaving only one path in the maze.

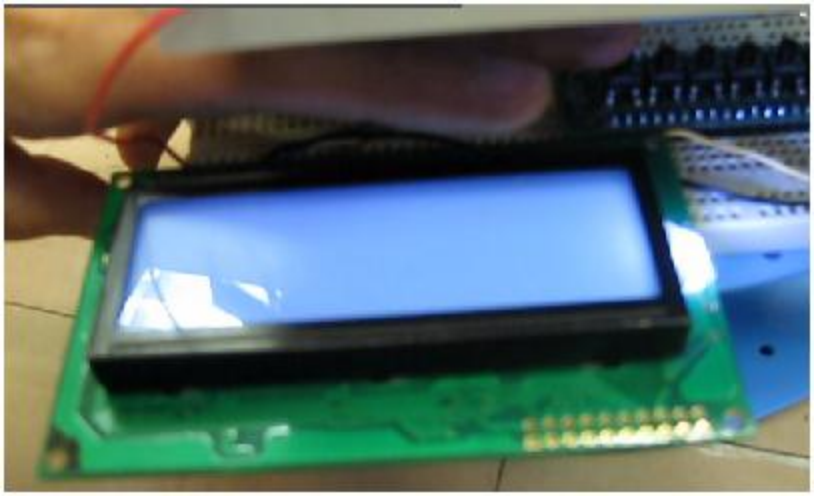
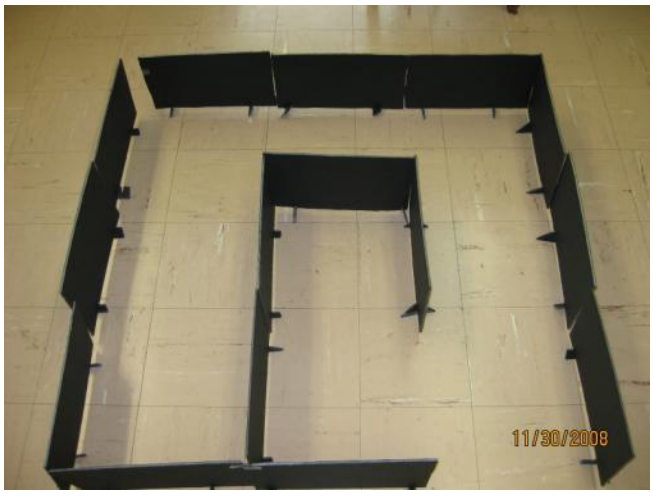


Figure 14 LCD Display and Switch

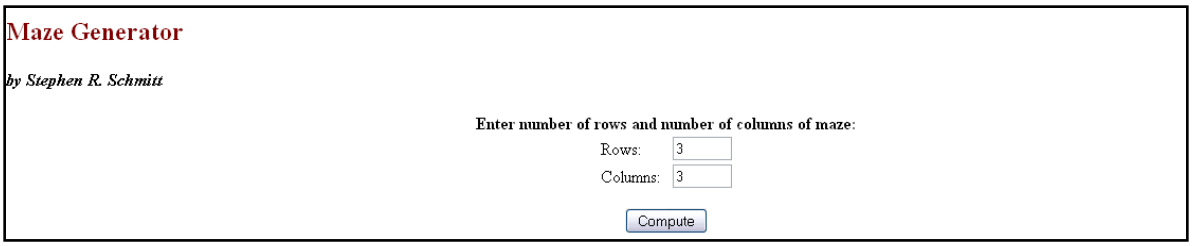
Last but not least, a maze must be constructed. The maze must meet several requirements. The path must be wide enough for the robot to traverse and turn. On the other hand, it must be narrow enough so that the sensors can make accurate detection. Ideally, the maze is "configurable," meaning that it is easy and fast to construct different typologies. After much experiment with materials and constructions, black foam, which is both light and firm, was adopted. Each block was cut to the carefully measured height and width.



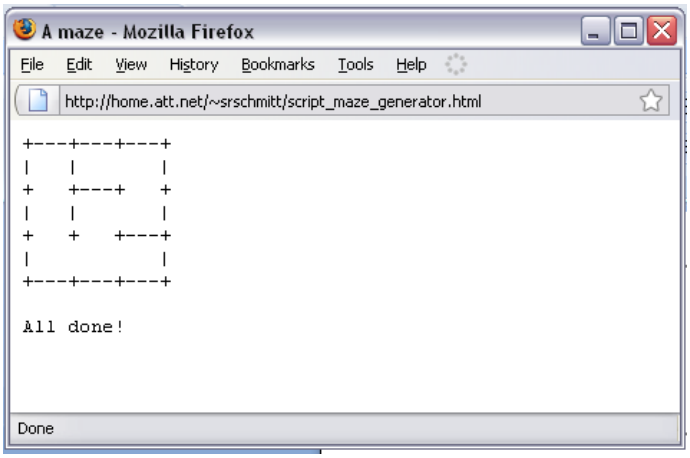
Software:

Maze-traversing and maze-solving algorithms are the most critical parts of implementing the desired functionalities. Much research was conducted on different types of mazes and maze-solving algorithms.

In the project proposal, the objective is narrowed down to solving perfect mazes with a wall-follower rule. A perfect maze, also known as "standard maze," is a maze containing no loops. It has one and only one path from any point in the maze to any other point. That is, the maze has no inaccessible sections, no circular paths, and no open areas. For this project, all the perfect mazes are generated with the help of *Maze Generator* online[14]. For example, to generate a 3x3 perfect maze, simply type in the sizes:



Compress compute, and a result is returned in a pop-up window:



[14] For more information, visit http://home.att.net/~srschmitt/script_maze_generator.html

A perfect maze is equivalent to a tree in graph theory. All the walls in the maze are connected together or to the maze's outer boundary.[15] It can be shown that all the walls can be deformed into a loop or circle, as shown below[16]

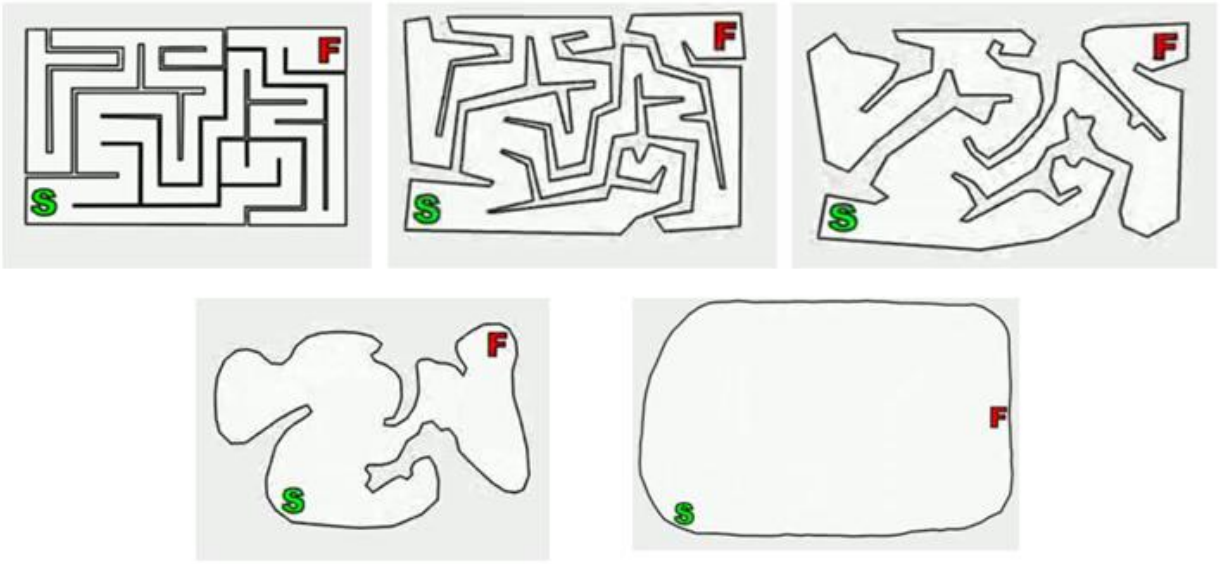


Figure 15 Perfect Maze Transformation

Obviously from the above graph, the robot is guaranteed to traverse every single location in the maze and return to the entrance by keeping one hand in contact with one wall of the maze. This is called the "Wall Follower" algorithm, also known as the left-hand rule or the right-hand rule. The wall following reduces to walking around a circle from start to finish.[17]

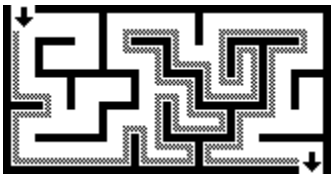


Figure 16 Traversal using right-hand rule

[15] In other words, the maze is *simply connected.*
[16] For more information, visit http://www.youtube.com/watch?v=k1tSK5V1pds
[17] If the maze is not simply connected, this method will not guarantee the robot to reach the exit.

The pseudo code below implements the right-hand rule:

```
DO
    GOSUB DETECT_ANY_OBSTACLES

    IF(no_obstacles_on_right) THEN
        GOSUB TURN_RIGHT
        GOSUB FORWARD
        GOSUB RECORD_CURRENT_LOCATION
        GOSUB RECORD_PATH
    ELSEIF(no_obstacles_in_front)
        GOSUB FORWARD
        GOSUB RECORD_CURRENT_LOCATION
        GOSUB RECORD_PATH
    ELSEIF(no_obstacles_on_left)
        GOSUB TURN_LEFT
        GOSUB FORWARD
        GOSUB RECORD_CURRENT_LOCATION
        GOSUB RECORD_PATH
    ELSE
        GOSUB TURN_AROUND
        GOSUB FORWARD
        GOSUB RECORD_CURRENT_LOCATION
        GOSUB RECORD_PATH
    ENDIF

    GOSUB CHECK_IF_DEST
WHILE(current_location != dest)

;when the switch is clicked
GOSUB DRAW_MAZE

;when the switch is clicked again
GOSUB SOLVE_MAZE
```

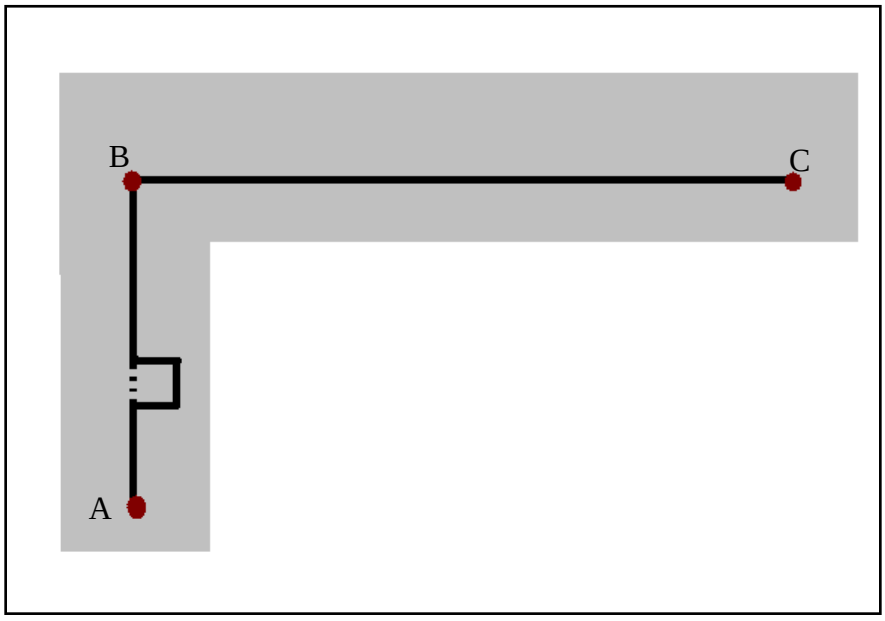
IV. RESULTS

After the robot is successfully assembled and tested, two videos were shot to demonstrate the robot meeting the objectives in Section 2.

A. Intelligent-Navigation

http://www.youtube.com/watch?v=cbX-Ss3xGvA&feature=related

In this video, the robot is tested on the parking lot across from Patrick Taylor Hall at Louisiana State University. The robot is given three GPS coordinates, A, B, and C, denoted by the red dots. The robot is instructed to start from A, turn right at B, and stop at C.

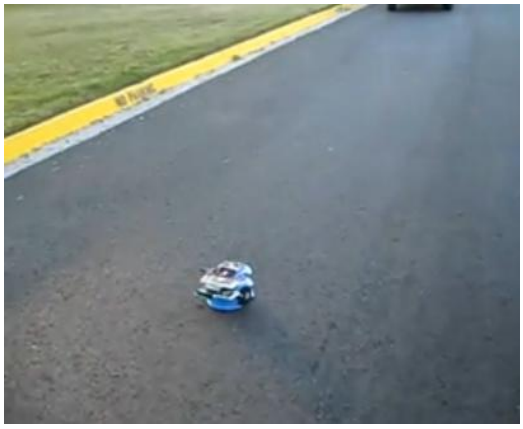


The robot successfully completed the task. It detected an obstacle on the path AB, and successfully avoided the obstacle. Further, it stayed on the specified path the entire time, automatically correcting its direction when it detected a deviation.

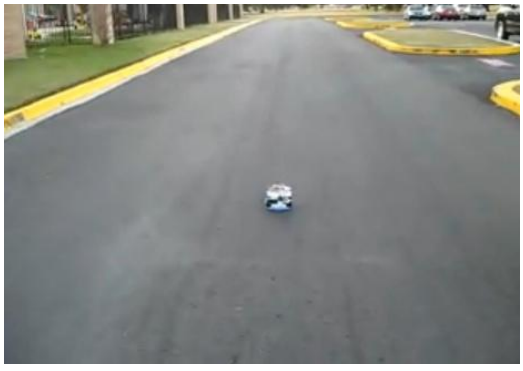
Some screenshots of the video are shown on the next page

1. The robot starts at GPS location A.

2. The robot detects and avoids a moving obstacle.



3. The robot turns right at GPS location B.

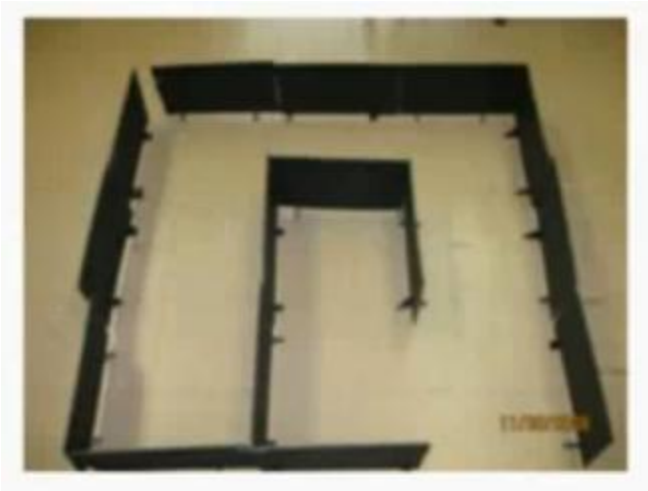


4. The robot stays in the middle of the road throughout and stops at GPS location C.

B. Maze-mapping and Maze-solving

http://www.youtube.com/watch?v=3GmwuYtv2HQ

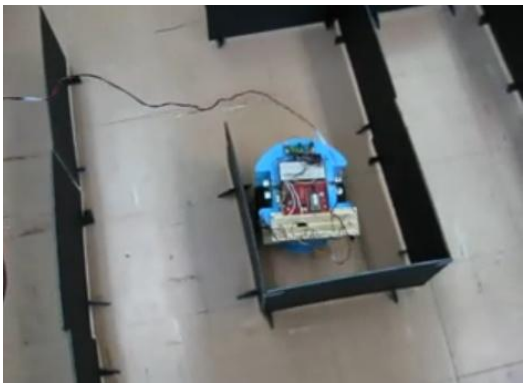
In this video, the robot is tested in a 3x3 perfect maze shown below



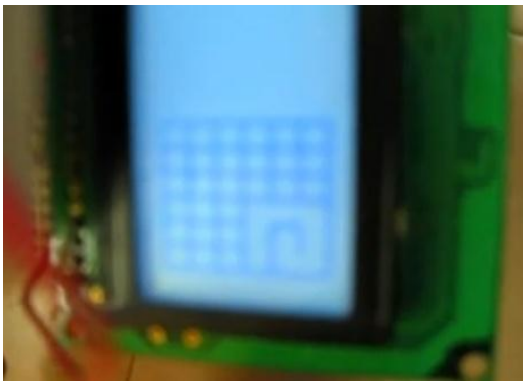
The robot travels from S to F using the right-hand rule and travels from F to S using the right-hand rule. Then the LCD display on the robot draws the traversed maze and solves it by darkening all dead ends. In the end, the robot travels from S to F while avoiding all dead ends.

The robot successfully completed all the tasks. Some screenshots of the video are shown below

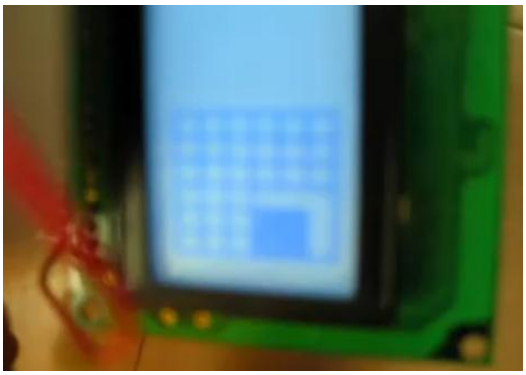
1. The robot travels from S to F with right-hand rule, making correct turns.



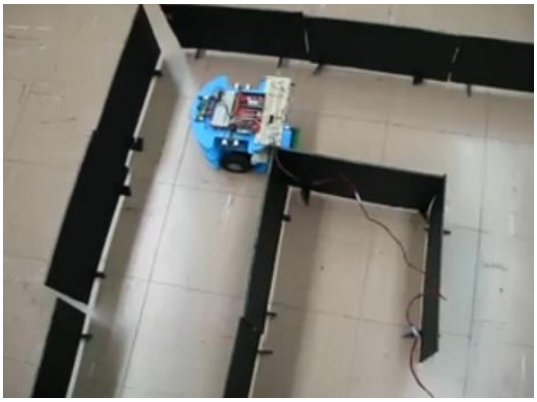
2. The robot travels from F to S with right-hand rule, making correct turns.



3. The robot draws the maze it just traversed on the LCD display.



4. The robot solves the maze by darkening the dead ends, leaving only one path.



5. The robot travels from S to F without using sensors and avoiding all dead ends.

V. CONCLUSION

In summary, this project consisted of designing, assembling, programming, interfacing, and testing a robot. The three main objectives for the robot were: intelligent navigation, maze solving, and maze mapping. After continuously working on the construction for over a semester, the robot was successfully completed, meeting all objectives originally set out.

This project proved to be an invaluable learning experience in my major field. "Computer engineering is a discipline that combines elements of both electrical engineering and computer science." Further, "Computer engineers are electrical engineers that have additional training in the areas of software design and hardware-software integration."[18] Indeed, this project has provided me opportunities to:

- Apply the core concepts learned from classes,
- Interface hardware and software to gain a comprehensive view,
- Get hands-on experience with a spectrum of electronic components,
- Become familiar with industry standards,
- Strengthen programming skills,
- Expand knowledge on algorithms, and
- Gain insights into various aspects of engineering practice.

[18] IEEE Computer Society; ACM (12 December 2004). *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. pg. iii. http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/CCCE-FinalReport-2004Dec12-Final.pdf. Retrieved on 2009-03-21.

With its powerful components, such as the agile motors, multi-purpose sensors, text and graphics display, accurate compass and GPS system, the robot can potentially be put into many applications. Moreover, the microprocessor can potentially host many elegant algorithms, making the robot "intelligent" for different applications. Last but not least, the hardware, software, and interfacing process prove to be very affordable, making the robot a cost-effective option for many types of jobs now and in the future.

REFERENCES

Calibrating the CMPS01, CMPS03 Robot Compass Modules. (n.d.). Retrieved March 23, 2009, from CMPS03 calibration documentation: http://litec.rpi.edu/Postings/cmps_cal.pdf

CMPS03 - Robot Compass Module. (n.d.). Retrieved March 23, 2009, from CMPS03 documentation: http://www.robotstorehk.com/CMPS03_release.pdf

GlobalSat. *GPS Receiver Engine Board EM-406A User Manual.* Taiwan: GlobalSat Technology Corporation .

Intelligent-Navigating Robot with Sensors, Compass, and GPS. (2008, December 3). Retrieved March 23, 2009, from YouTube: http://www.youtube.com/watch?v=cbX-Ss3xGvA&feature=related

Lynxmotion. *IRPD Ver 3.0 Infrared Proximity Detector.* Pekin: Lynxmotion, Inc .

MatrixOrbital. (2008). *GLK12232-25-WBL Technical Manual.* Calgary: MatrixOrbital.

mazemaster225. (2007, December 23). *Maze to Tree.* Retrieved March 23, 2009, from YouTube: http://www.youtube.com/watch?v=k1tSK5V1pds

Parallax. (2008). *PING))) Ultrasonic Sensor* . Retrieved March 23, 2009, from www.parallax.com: http://www.parallax.com/Store/Sensors/ObjectDetection/tabid/176/ProductID/92/List/1/Default.aspx?SortField=ProductName,ProductName

Schmitt, S. R. (2004). *Maze Generator* . Retrieved March 23, 2009, from Maze Generator : http://home.att.net/~srschmitt/script_maze_generator.html

IEEE Computer Society. (2006). *Computer Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering.* Retrieved 3 21, 2009, from ACM: http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/CCCE-FinalReport-2004Dec12-Final.pdf

Seattle Robotics Society. (2009). *Whats a servo: A quick tutorial*. Retrieved March 23, 2009, from Seattle Robotics Society : http://www.seattlerobotics.org/guide/servos.html

Wikipedia. (2009). *Maze - Wikipedia, the free encyclopedia*. Retrieved March 23, 2009, from Wikipedia: http://en.wikipedia.org/wiki/Maze

Yuan, Y. (2008). *EEProject.wmv.* Retrieved March 23, 2009, from YouTube: http://www.youtube.com/watch?v=3GmwuYtv2HQ&feature=channel_page

APPENDIX

Appendix A: Code for Intelligent Navigation

```
Main
LCD          CON 0
COMPASS              CON 1
GPS                  CON 2
ButtonPin  CON 3
FrontPingR CON 4
FrontPingL CON 5
LeftMotor  CON 6
RightMotor CON 7

Forever1  VAR BYTE
Forever2  VAR BYTE
Work1     VAR BYTE

COORDINATES VAR WORD(48)
COUNTER                  VAR BYTE
TIME1                VAR WORD
TIME2                VAR WORD
COORD                VAR WORD
iniSignal  VAR WORD
signal               VAR WORD
direction  VAR WORD
direction2  VAR WORD
location   VAR WORD
counter2  VAR WORD
TooRight VAR BYTE
TooLeft              VAR BYTE
Should_Turn          VAR BIT
i                        VAR BYTE

FrontR       VAR WORD
FrontL       VAR WORD


SEROUT LCD,N19200,[254,88]
        Forever1 = 0

        ;//////////////Go until making the first turn /////////////////
        DO
                FOR COUNTER = 0 TO 100
                        GOSUB FORWARD_FAST
                        GOSUB DETECT
                        IF(FrontR=0)AND(FrontL=0)THEN
                                GOSUB AVOID
                        ENDIF
                NEXT

                TooRight = 0
          TooLeft = 0
          FOR COUNTER = 0 TO 3
                        PULSIN COMPASS,1,signal
                        IF(signal<13850) THEN
                                TooLeft = TooLeft+1
                        ELSEIF(signal>13860)
```

```
                    TooRight = TooRight + 1
                    ENDIF
            NEXT
            IF(TooLeft>2) THEN
                    GOSUB ADJUST_RIGHT
            ELSEIF(TooRight>2)
                    GOSUB ADJUST_LEFT
            ENDIF


            SERIN GPS,N4800,[WAIT("$GPGGA"),str COORDINATES\47]
            Should_Turn = 0
            GOSUB CHECK_TURN1

    WHILE(Should_Turn=0)

    ;//////////////Turn right //////////////////
    PAUSE 3000
    GOSUB TURN_RIGHT90

    ;//////////////Go until making the second turn
    PAUSE 3000
    DO
            FOR COUNTER = 0 TO 100
                    GOSUB FORWARD_FAST
                    GOSUB DETECT
;                   IF(FrontR=0)AND(FrontL=0)THEN
;                           GOSUB AVOID
;                   ENDIF
            NEXT
            TooRight = 0
       TooLeft = 0
       FOR COUNTER = 0 TO 3
                    PULSIN COMPASS,1,signal
                    IF(signal<24620) THEN
                            TooLeft = TooLeft+1
                    ELSEIF(signal>24650)
                    TooRight = TooRight + 1
                    ENDIF
            NEXT
            IF(TooLeft>2) THEN
                    GOSUB ADJUST_RIGHT
            ELSEIF(TooRight>2)
                    GOSUB ADJUST_LEFT
            ENDIF



            SERIN GPS,N4800,[WAIT("$GPGGA"),str COORDINATES\47]
            Should_Turn = 0
            GOSUB CHECK_TURN2

    WHILE(Should_Turn=0)
    ;//////////////Turn right //////////////////
    PAUSE 3000
    GOSUB TURN_LEFT90
```

```
        ;////////////Walk for a little bit/////////
        FOR COUNTER = 1 TO 5
                GOSUB GO_SHORT
        NEXT


END

FORWARD_FAST:
        LOW LeftMotor
        LOW RightMotor
        PULSOUT LeftMotor, 3340
        PULSOUT RightMotor, 10
RETURN

CHECK_TURN1:
   location = 0
        location = location + (COORDINATES(17)-30)*1000
        location = location + (COORDINATES(18)-30)*100
        location = location + (COORDINATES(19)-30)*10
        location = location + (COORDINATES(20)-30)
        SEROUT LCD,N19200,[254,71,0,0]
        SEROUT LCD,N19200,["Location: "]
        SEROUT LCD,N19200,[254,71,1,0]
        SEROUT LCD,N19200,[DEC location]
        if (location < 22940) THEN
     Should_Turn = 1;
     SEROUT LCD,N19200,["TURN HERE"]
        endif

RETURN

CHECK_TURN2:
   location = 0
        location = location + (COORDINATES(30)-30)*1000
        location = location + (COORDINATES(31)-30)*100
        location = location + (COORDINATES(32)-30)*10
        location = location + (COORDINATES(33)-30)

        SEROUT LCD,N19200,[254,88]
        SEROUT LCD,N19200,[254,71,0,0]
        SEROUT LCD,N19200,[DEC location]
        IF (location > 28190) THEN
     Should_Turn = 1
     SEROUT LCD,N19200,["TURN HERE"]
        ENDIF
RETURN

ADJUST_RIGHT:
        ;PULSIN COMPASS, 1, signal
        ;IF(signal < iniSignal-40) THEN
        ;       Do
        ;               PULSIN COMPASS,1,signal
        ;               PAUSE 10
```

```
                        FOR i=1 to 5
                        LOW RightMotor
                        LOW LeftMotor
                        PULSOUT LeftMotor, 3000
                        PULSOUT RightMotor, 1525

                        NEXT
        ;       WHILE (signal <iniSignal-20)
        ;ENDIF
RETURN

ADJUST_LEFT:
        ;PULSIN COMPASS, 1, signal
        ;IF(signal > iniSignal+40) THEN
        ;       Do
        ;               PULSIN COMPASS,1,signal
        ;               PAUSE 10

                        FOR i=1 to 20
                        LOW RightMotor
                        LOW LeftMotor
                        PULSOUT LeftMotor, 1894
                        PULSOUT RightMotor, 45
                        NEXT

        ;       WHILE (signal > iniSignal+20)
        ;ENDIF
RETURN



DISPLAY:
        Forever2=0
        DO
        SERIN GPS,N4800,[WAIT("$GPGGA"),str COORDINATES\47]
;       SEROUT LCD,N19200,[str COORDINATES\45]
        SEROUT LCD,N19200,[254,71,0,0]
        SEROUT LCD,N19200,["TIME: "]
        GOSUB CHECK_TURN1
        FOR COUNTER= 0 TO 2
                TIME1 = COORDINATES(COUNTER*2+1)
                TIME2 = COORDINATES(COUNTER*2+2)
                SEROUT LCD,N19200,[TIME1]
                SEROUT LCD,N19200,[TIME2]
                SEROUT LCD,N19200,[":"]
  NEXT
        SEROUT LCD,N19200,[254,71,0,2]
        SEROUT LCD,N19200,["LAT: "]
        FOR COUNTER= 12 TO 22
                SEROUT LCD,N19200,[COORDINATES(COUNTER)]
  NEXT
        SEROUT LCD,N19200,[254,71,0,3]
        SEROUT LCD,N19200,["LONG: "]
        FOR COUNTER= 24 TO 35
                SEROUT LCD,N19200,[COORDINATES(COUNTER)]
  NEXT
```

```
   ;PAUSE 10
   PULSIN COMPASS,1,signal
            ;SEROUT LCD, N19200, ["DIR:", DEC signal, " Degrees", 13]
            IF (signal=0) THEN
              direction = direction
            ELSEIF (((signal <1000) OR (signal > 35000)))
              direction = "N"
              direction2 = " "
            ELSEIF ((signal >1000) AND (signal < 5000))
              direction = "N"
              direction2 = "E"
   ELSEIF ((signal < 5500) AND (signal>5000))
              direction = "E"
              direction2 = " "
              ELSEIF ((signal >5500) AND (signal < 10500))
              direction = "S"
              direction2 = "E"
            ELSEIF (signal<11500) AND (signal>10500)
              direction = "S"
              direction2 = " "
              ELSEIF ((signal >11500) AND (signal < 17500))
              direction = "S"
              direction2 = "W"
            ELSEIF (signal<18500) AND (signal>17500)
              direction = "W"
              direction2 = " "
              ELSEIF ((signal >18500) AND (signal < 35000))
              direction = "N"
              direction2 = "W"
            ENDIF
;           PAUSE 10
            SEROUT LCD,N19200,[254,71,0,4]
            SEROUT LCD,N19200,["           "]
            SEROUT LCD,N19200,[254,71,0,4]
            SEROUT LCD,N19200,["DIR: "]
            SEROUT LCD,N19200,[direction, direction2, " ", DEC signal, "deg."]
            WHILE(Forever2=0)
RETURN

DETECT:
  HIGH FrontPingR
  pause 10
  FrontR = in12
  LOW FrontPingR

  HIGH FrontPingL
  pause 10
  FrontL = in12
  LOW FrontPingL

RETURN

AVOID:
SEROUT LCD,N19200,[254,71,0,0]

              PAUSE 1000
```

```
            GOSUB TURN_RIGHT90

            PAUSE 100
            for i=1 to 4
                    GOSUB GO_SHORT
            next

            PAUSE 100
            GOSUB TURN_LEFT90
            PAUSE 100
            for i=1 to 4
                    GOSUB GO_SHORT
            next

            PAUSE 100
            GOSUB TURN_LEFT90
            PAUSE 100
            for i=1 to 4
                    GOSUB GO_SHORT
            next

            PAUSE 100
            GOSUB TURN_RIGHT90

            SEROUT LCD,N19200,[254,71,0,0]
          SEROUT LCD,N19200,[" GETTING OUT OF AVOID "]
          PAUSE 1000

RETURN


TURN_LEFT90:

          FOR i = 1 to 67
                    LOW Leftmotor
                    LOW Rightmotor
                    PULSOUT Leftmotor, 1894
                    PULSOUT Rightmotor, 45
                    PAUSE 10
          NEXT
          RETURN

TURN_RIGHT90:
                    SEROUT LCD,N19200,["Inside turn_right90 "]

          FOR i = 1 to 160
                    LOW LeftMotor
                    LOW RightMotor
                    PULSOUT LeftMotor, 3000
                    PULSOUT RightMotor, 1525
          NEXT

          RETURN

GO_SHORT:
;         pause 10000
```

```
        counter2 = 100
        Do
        GOSUB FORWARD_SLOW
        counter2 = counter2 -1
        WHILE counter2 <>0
;       pause 10000
RETURN

FORWARD_SLOW:

        ;FOWWARD SLOW
        LOW LeftMotor
        LOW RightMotor
        PULSOUT LeftMotor, 2029
        PULSOUT RightMotor, 1391
        PAUSE 10
RETURN
```

Appendix B: Code for Maze Solving and Mapping

```
MAIN
LCD          CON 0
ButtonPin  CON 1
RightPing  CON 2
LeftPing   CON 3
FrontPingR CON 4
FrontPingL CON 5
LeftMotor  CON 6
RightMotor CON 7

Trigger  CON 1              ' 10 uS trigger pulse
IsHigh   CON 1               ' for PULSOUT
IsLow    CON 0

;Sensor Variables
RightRawDist VAR WORD
LeftRawDist  VAR WORD
Prev_Dist VAR WORD
FrontR      VAR WORD
FrontL      VAR WORD
Right               VAR WORD
Left                VAR WORD
Counter     VAR WORD

i                       VAR WORD
work1               VAR BYTE
forever         VAR BYTE

;Mapping Variables
CurrentDir          VAR BYTE
CurrentBlk   VAR BYTE
SidesArray          VAR BYTE(73)
DeadEndArray VAR BYTE(36)

;Display Variables
```

```
Index                   VAR BYTE
Blk                             VAR BYTE
X1                              VAR BYTE
X2                              VAR BYTE
Y1                              VAR BYTE
Y2                              VAR BYTE
CheckNext    VAR BIT
NoDeadEnds   VAR BIT

 ;Initialize variables
 CurrentDir = 0
 CurrentBlk = 1
 FOR Counter= 1 to 72
          SidesArray(Counter) = 1
 NEXT

 FOR Counter= 1 to 72
          DeadEndArray(Counter) = 1
 NEXT

 forever = 0
 DO
 BUTTON ButtonPin,1,0,2,work1,1,STEP1
 WHILE(forever=0)

STEP1:

 PAUSE 1000
 GOSUB GET_ALL_SIDES

 IF(RightRawDist>300) THEN
   GOSUB TURN_RIGHT
   PAUSE 100
   GOSUB FORWARD
   PAUSE 100
   CurrentDir = (CurrentDir+1)//4
   SidesArray(CurrentBlk*2-2) = 0

 ELSEIF (FrontR<>0)OR(FrontL<>0)
   GOSUB FORWARD
   PAUSE 100
   CurrentDir = CurrentDir
   SidesArray(CurrentBlk*2-2) = 1
   SidesArray(CurrentBlk*2-1) = 0

 ELSEIF (LeftRawDist > 300)
   GOSUB TURN_LEFT
   PAUSE 100
   GOSUB FORWARD
   PAUSE 100
   CurrentDir = (CurrentDir+3)//4
   SidesArray(CurrentBlk*2-2) = 1
   SidesArray(CurrentBlk*2-1) = 1
   SidesArray(CurrentBlk*2) = 0

 ELSE
```

```
    GOSUB TURN_AROUND
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+2)//4
    SidesArray(CurrentBlk*2-2) = 1
    SidesArray(CurrentBlk*2-1) = 1
    SidesArray(CurrentBlk*2) = 1
  ENDIF

  IF(CurrentDir=0) THEN
            CurrentBlk = CurrentBlk + 6
  ELSEIF(CurrentDir=1)
            CurrentBlk = CurrentBlk - 1
  ELSEIF(CurrentDir=2)
            CurrentBlk = CurrentBlk - 6
  ELSE
            CurrentBlk = CurrentBlk + 1
  ENDIF

  IF(CurrentBlk = 15) THEN
            GOSUB GET_ALL_SIDES
            IF(RightRawDist>300) THEN
    GOSUB TURN_RIGHT
    PAUSE 100
            ENDIF
            GOTO STEP1_DONE
  ENDIF

GOTO STEP1

STEP1_DONE:

  forever = 0
  DO
  BUTTON ButtonPin,1,0,2,work1,1,STEP2
  WHILE(forever=0)

STEP2:
  PAUSE 1000
  GOSUB GET_ALL_SIDES

  IF(RightRawDist>300) THEN
    GOSUB TURN_RIGHT
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+1)//4
    SidesArray(CurrentBlk*2-2) = 0

  ELSEIF (FrontR<>0)OR(FrontL<>0)
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = CurrentDir
    SidesArray(CurrentBlk*2-2) = 1
    SidesArray(CurrentBlk*2-1) = 0
```

```
  ELSEIF (LeftRawDist > 300)
    GOSUB TURN_LEFT
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+3)//4
    SidesArray(CurrentBlk*2-2) = 1
    SidesArray(CurrentBlk*2-1) = 1
    SidesArray(CurrentBlk*2) = 0

  ELSE
    GOSUB TURN_AROUND
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+2)//4
    SidesArray(CurrentBlk*2-2) = 1
    SidesArray(CurrentBlk*2-1) = 1
    SidesArray(CurrentBlk*2) = 1
  ENDIF

  IF(CurrentDir=0) THEN
            CurrentBlk = CurrentBlk + 6
  ELSEIF(CurrentDir=1)
            CurrentBlk = CurrentBlk - 1
  ELSEIF(CurrentDir=2)
            CurrentBlk = CurrentBlk - 6
  ELSE
            CurrentBlk = CurrentBlk + 1
  ENDIF

  IF(CurrentBlk = 1)AND(CurrentDir=1) THEN
           GOTO STEP2_DONE
  ENDIF

GOTO STEP2

STEP2_DONE:
  forever = 0
  DO
  BUTTON ButtonPin,1,0,2,work1,1,STEP3
  WHILE(forever=0)

STEP3:
  GOSUB DISPLAY_MAZE
  GOSUB DARKEN_DEADENDS
  GOTO STEP3_DONE

STEP3_DONE:
  forever = 0
  DO
  BUTTON ButtonPin,1,0,2,work1,1,STEP4
  WHILE(forever=0)

STEP4:
```

```
  CurrentBlk = 1
  IF(SidesArray(CurrentBlk*2-2)=0) THEN
    GOSUB TURN_RIGHT
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+1)//4

  ELSEIF (SidesArray(CurrentBlk*2-1)=0)
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = CurrentDir

  ELSEIF (SidesArray(CurrentBlk*2)=0)
    GOSUB TURN_LEFT
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+3)//4

  ELSE
    GOSUB TURN_AROUND
    PAUSE 100
    GOSUB FORWARD
    PAUSE 100
    CurrentDir = (CurrentDir+2)//4
  ENDIF

  IF(CurrentDir=0) THEN
            CurrentBlk = CurrentBlk + 6
  ELSEIF(CurrentDir=1)
            CurrentBlk = CurrentBlk - 1
  ELSEIF(CurrentDir=2)
            CurrentBlk = CurrentBlk - 6
  ELSE
            CurrentBlk = CurrentBlk + 1
  ENDIF

  IF(CurrentBlk = 15) THEN
    SEROUT LCD,N19200,[254,71,0,0]
            SEROUT LCD,N19200,["MISSION COMPLETED"]
            END
  ENDIF

GET_ALL_SIDES:
  counter = 0
  DO
            HIGH FrontPingR
            pause 10
            FrontR = in12
            LOW FrontPingR

            HIGH FrontPingL
            pause 10
            FrontL = in12
            LOW FrontPingL
```

```
            counter = counter+1
  WHILE(counter<10)AND(FrontR=0)AND(FrontL=0)

  RightRawDist = 0
  DO
              Prev_Dist = RightRawDist
              LOW RightPing
              PAUSE 2
              PULSOUT RightPing, Trigger
              PULSIN  RightPing, IsHigh, RightRawDist
              RightRawDist = RightRawDist / 2
  WHILE (abs(Prev_Dist-RightRawDist)>50)

  LeftRawDist = 0
  DO
     Prev_Dist = RightRawDist
              LOW LeftPing
              PAUSE 2
              PULSOUT LeftPing, Trigger
              PULSIN  LeftPing, IsHigh, LeftRawDist
              LeftRawDist = LeftRawDist / 2
  WHILE (abs(Prev_Dist-RightRawDist)>50)
RETURN

TURN_AROUND:
              FOR Counter= 1 to 52
              LOW LeftMotor
              LOW RightMotor
              PULSOUT LeftMotor, 3000
              PULSOUT RightMotor, 1525
              pause 30
              NEXT
RETURN

TURN_LEFT:
              Counter = 380
              Do
              LOW LeftMotor
              LOW RightMotor
              PULSOUT LeftMotor, 1894
              PULSOUT RightMotor, 45
              Counter = Counter -1
              WHILE Counter <>0
RETURN


TURN_RIGHT:
              ;go forward for a while
              Counter = 27
              Do
              LOW LeftMotor
              LOW RightMotor
              PULSOUT LeftMotor, 3000
              PULSOUT RightMotor, 1525
              pause 30
              Counter = Counter -1
```

```
        WHILE Counter <>0
RETURN

FORWARD:
        FOR counter = 0 to 650
                GOSUB FORWARD_FAST

                IF(counter//10=0)THEN
                        GOSUB GET_SIDES
                        IF(RightRawDist<100) THEN
                                GOSUB TURN_SLIGHT_LEFT
                        ELSEIF(LeftRawDist<100)
                                GOSUB TURN_SLIGHT_RIGHT
                        ENDIF
                ENDIF
        NEXT
RETURN

GET_SIDES:

  LOW RightPing
  PAUSE 2
  PULSOUT RightPing, Trigger
  PULSIN  RightPing, IsHigh, RightRawDist
  RightRawDist = RightRawDist / 2

  LOW LeftPing
  PAUSE 2
  PULSOUT LeftPing, Trigger
  PULSIN  LeftPing, IsHigh, LeftRawDist
  LeftRawDist = LeftRawDist / 2

RETURN

FORWARD_FAST:
        LOW LeftMotor
        LOW RightMotor
        PULSOUT LeftMotor, 3340
        PULSOUT RightMotor, 10
RETURN

TURN_SLIGHT_RIGHT:
        ;==========
        FOR i=1 to 2
        LOW RightMotor
        LOW LeftMotor
        PULSOUT LeftMotor, 3000
        PULSOUT RightMotor, 1525
        NEXT
        ;PAUSE 100
RETURN

TURN_SLIGHT_LEFT:
        ;==========
        FOR i=1 to 2
        LOW RightMotor
```

```
        LOW LeftMotor
        PULSOUT LeftMotor, 1894
        PULSOUT RightMotor, 45
        NEXT
;       PAUSE 500
RETURN

DISPLAY_MAZE:
        SEROUT LCD,N19200,[254,108,120,0,90,0]
        SEROUT LCD,N19200,[254,108,120,0,120,30]

        Index = 1
        Do

                IF(SidesArray(Index) = 1)THEN
                        Blk = (Index+1)/2
                        X1 = 120-((Blk-1)/6+1)*5
                        Y1 = ((Blk-1)//6)*5
                        X2 = X1
                        Y2 = Y1+5
                        SEROUT LCD,N19200,[254,108,X1,Y1,X2,Y2]
                ENDIF

        Index = Index+1
        IF(SidesArray(Index) = 1)THEN
                        Blk = Index/2
                        X1 = 120-((Blk-1)/6+1)*5
                        Y1 = ((Blk-1)//6+1)*5
                        X2 = X1+5
                        Y2 = Y1
                        SEROUT P0,N19200,[254,108,X1,Y1,X2,Y2]
                ENDIF

                Index = Index+1
                PAUSE 100

        WHILE Index <72
RETURN

DARKEN_DEADENDS:
        DO
                FOR Index = 1 to 36
                        IF(SidesArray(Index*2)+SidesArray(Index*2-1)+SidesArray(Index*2-2)+SidesArray(Index*2-
13)=3)THEN
                                Blk = (Index+1)/2
                                X1 = 120-((Blk-1)/6+1)*5
                                Y1 = ((Blk-1)//6)*5
                                X2 = X1+5
                                Y2 = Y1+5
                                SEROUT LCD,N19200,[254,120,1,X1,Y1,X2,Y2]

                                SidesArray(Index*2) = 1
                                SidesArray(Index*2-1) = 1
                                SidesArray(Index*2-2) = 1
                                SidesArray(Index*2-13) = 1
```

```
                                        DeadEndArray(Index) = 1
                                        Index = Index+1
                                        PAUSE 100
                                ENDIF
                        NEXT
                        GOSUB CHECK_DEADENDS
                WHILE(NoDeadEnds=0)
RETURN

CHECK_DEADENDS:
                NoDeadEnds = 1
                FOR Index = 1 to 36
                        IF(DeadEndArray(Index)=0)AND(SidesArray(Index*2)+SidesArray(Index*2-1)+SidesArray(Index*2-
2)+SidesArray(Index*2-13)=3)THEN
                                NoDeadEnds = 0
                        ENDIF
                NEXT
RETURN
```