1. INTRODUCTION

Robots are capable of performing many different tasks and operations precisely and do not require common safety and comfort elements that humans need. However, it takes much effort and many resources to make a robot function properly. Robots are used in the auto, medical and manufacturing industries, in all manners of exploration vehicles, and, of course in many science fiction films. The word “robot” first appeared in a Czechoslovakian satirical play, Rossum Universal Robots, by Karel Capek in 1920. Robot in this play tended to be human-like. From this point onward it seemed that many science fiction stories involved these robots trying to fit into society and make sense out of human emotion. This changed when General Motors installed the first robots in their manufacturing plants in 1961. These automated machines presented an entirely different image from the “human form” robots of science fiction [1].

Building and programming a robot is a combination of mechanics, electronics, and problem solving. The mechanical principles, example program listings, and circuits that were used in this thesis are very similar to industrial applications developed by engineers.

The work in robotics make the worker interested and excited about the fields of engineering, mecatronics, and software development as they design, construct, and program an autonomous robot [2] .

Mobile robots pose a unique challenge to artificial intelligence researchers. They are inherently autonomous, and they need to deal with key issues such as uncertainty in sensing and action, reliability, and real-time response. Mobile robots also require the integration of sensing, acting and planning within a single system. These are all hard problems, but ones that must be solved if we are to have a truly autonomous, intelligent system.

The navigation of a mobile robot through an environment with obstacles is based on the knowledge of the vehicle's current position. Position estimation is a very important capability for a mobile robot in order to perform a useful mission.

The robot's odometric system provides fast positional information, but it involves unavoidable errors that can increase positional uncertainty. Therefore, by using dead reckoning techniques and the vehicles kinematics model, it is possible to produce an estimation of the current position. This method has high performance for short distances and can be computed very quickly, so it uses the vehicle's real time controller.

However, because these estimations are not free of errors, a higher level system is necessary to periodically reduce the position uncertainty by using external sensors. Position uncertainty reduction techniques are very time expensive; therefore they must only be used when the uncertainty grows to dangerous levels.

Some of the most difficult problems or needs that arise when developing an autonomous mobile robot are path tracking, navigation and perception [3]. Robot path tracking and obstacle avoidance can be applied in the entire environment, but it is important in hazardous environments such as underwater, space, and remote locations that are dangerous for humans.

In this thesis the path tracking and navigation problems of mobile robot are considered. As an example, the "Boe-Bot" robot is considered. Its name comes from the board of education carrier board that is mounted on its wheeled chassis. There are different methods for navigating Boe-Bot robots are: basic Boe-Bot navigation under program control, navigation using a variety of sensor input, navigation using feedback and various control techniques, and navigation using artificial intelligence.

The aim of the thesis is the development of software for path tracking and navigation of mobile Boe-Bot robot. For software development a Parallax system with Basic Stamp language is used. Thesis includes four chapters, conclusion, references and appendices.

In order to achieve the aims of this thesis the following work was done:

- Collecting the major components of the Boe-Bot robot
- Calibrating the servo motors of the robot and centering them
- Building and testing the needed circuit for whiskers, photoresistor, and IR
- Designing and implementing the programs for obstacle avoidance (using different sensors) and path tracking
- Simulating and running the system to obtain practical results
- Analyzing the reasons of differences between these results

In Chapter 2, a brief discussion of obstacle avoidance and path tracking problems of mobile robots is given. Essential requirements for robot obstacle avoidance and path tracking are discussed.

In Chapter 3, the hardware and software of the Boe-Bot Robot are considered. The navigation of the Boe-Bot robot to perform a variety of maneuvers and ramping is considered. Also different types of sensors (whisker, photoresistor, infrared) that are used in navigation of the Boe-Bot robot are described. The information measurement system of robot is described. The robot firstly detects distance using a frequency sweep technique by its infrared sensors. The photoresistor measures responsively shadow and this information is used in controlling the Boe-Bot robot during path tracking. The color of line that the robot were to track is defined by measuring RC decay time giving by photoresister sensor. The algorithms for calculating distance and angle for goal navigation and path tracking were written.

In Chapter 4, the structure of the obstacle avoidance system is described, and flow charts of obstacle avoidance and path tracking algorithms are given. Lastly the simulation of obstacle avoidance and path tracking of mobile robot were conducted.

Conclusion includes important results obtained from the work reported in this thesis.

CHAPTER TWO. PATH TRACKING AND NAVIGATION USING MOBILE ROBOTS

2.1 Obstacle avoidance and path tracking of mobile robots

Path tracking and obstacle avoidance are two very important behaviors that must be considered in the process of developing Autonomous Ground Vehicles (AGV) or mobile robots. An AGV is an unmanned vehicle with the ability to operate all by itself at ground level under the control of a computer. A lot of progress has been done in the field of developing AGVs and mobile robots in the last decade, and successful applications have been made in both mining and agriculture.

Obstacle avoidance and path tracking of mobile robots is a broad topic, covering a large spectrum of different technologies and applications. It draws on some very ancient techniques, as well as some of the most advanced space science and engineering. Path planning of mobile robots, in particular the case where the environment is known, is a well studied problem, one problem is that often no complete knowledge about the environment is available. Having a detailed map with all the obstacles marked seems to be unrealistic for most situations. In many outdoor applications, the robots can determine their coordinates by using, for example, GPS. However the knowledge about the surroundings may often be very limited. Under such conditions, there is too much uncertainty for a very detailed plan to make sense. For preplanning purposes, a coarser choice is probably good enough. Additionally it is important to be able to replan the path online based on new information obtained by sensors while navigating. A natural way of updating plans is to first select a path based on the present knowledge, then move along that path for a short time while collecting new information. Based on the new findings the path is then replanned. [13, 14].

This methodology is often used in the literature for path planning in unknown areas. One of the original motivations for studying this problem was the terrain acquisition problem, where a robot is required to produce a complete map of an unknown terrain. In many publications, graph methods are used for solving the path planning task.

The key difference between robotic and human navigation is the quantum difference in perceptual capabilities. Humans can detect, classify, and identify environmental features under widely varying environmental conditions, independent of relative orientation and distance. Current robots, while being able to detect stationary obstacles before they run into them, have very limited perceptual and decisional capabilities. Although much research is being done to improve robotic navigational performance through enhanced perception, decisions to utilize these emerging technologies must be based on a critical analysis of considerations of technical risk and cost [4].

Obstacle avoidance is a fundamental problem of robot navigation. In general, robotic navigation involves three distinct aspects:

- **Global** navigation, which is the ability to determine one's position in absolute or map-referenced terms, and to move to a desired destination point.
- **Local** navigation, the ability to determine one's position relative to objects (stationary or moving) in the environment, and not collide with them as one move.
- **Personal** navigation, which involves being aware of the positioning of the various parts that make up oneself, in relation to each other and handling objects.

High speed navigation is preferable in order to achieve service efficiencies. There are fundamental difficulties when we want to increase the speed of a mobile robot. Such problems can be classified into three categories as follows:

1) Dynamic and mechanical limitations
2) Control and computational limitations
3) Unexpected dynamic changes in the environment

The first problem: implies that there might take place wheel slippage or rollover of the robot when excessive speed is applied when the robot makes a sharp cornering or an emergency stop. This problem can be solved by appropriate modeling of a mobile robot

dynamics. In practical applications, the first problem is rarely considered, because other problems provide more strict limitation on the maximum speed of the mobile robot [5].

The second problem: can be interpreted as a real-time obstacle avoidance problem. The speed of navigation can be limited by sensor capabilities to detect obstacles, sensing speed, computational cost of the obstacle avoidance algorithms, and motion control response. There has been a lot of research on the dynamic obstacle avoidance problem. A mobile robot can navigate real environments without collision by adopting some useful developed technologies. Owing to the fast computational speed of recent CPUs, a robot's motion can be controlled with acceptably high update rate [6].

The third problem: In order to deal with unexpected dynamic changes of the environment, a robot should utilize its own experiences. Humans fully exploit their experiences in real environment in many cases. Suppose that a person is walking in corridor. He might walk fast when there is no obstacle. He might reduce the walking speed when he expects that another person possibly burst into the corridor through the door from a room. Alternatively, a person might reduce the speed when he already knows that a part of the floor is slippery. This fact implies that a person possibly changes walking speed even though there are no visible obstacles. In the presented case, a person should have a location-dependant preliminary knowledge of the environment for controlling the walking speed.

We focus on high speed navigation without collision with unexpected dynamic obstacles in corridor environment. The human coexisting environment has the two types of dynamic obstacles to cope with. The first is the expected dynamic obstacle which can be detected by sensors. The second is unexpected dynamic obstacles that abruptly emerge. Although current state-of-the-art solutions solved the problem of the safe and fast navigation against the expected dynamic obstacle, it is still the difficult to be solved for the case of the unexpected dynamic obstacle. The objective is to achieve safe and fast navigation for the both cases of dynamic obstacle. Because the dynamic obstacle avoidance is one of the main issues for the robotics researcher, there are many ongoing researches about controlling robot behavior avoiding the obstacles. The approaches can be classified into three categories**.** One is model based path planning. Another is sensor-

based reactive motion control. The other is a hybrid approach which combines two schemes.

A model based path planning uses models of the world and robot to compute a path for the robot to reach its goal. Although the motion of the robot can be obtained in a quite simple way, it is difficult to use the original potential field due to a local minima problem. The gradient method provides a global optimal solution for the path planning problem. However, it is still difficult to be applied to the dynamic obstacle avoidance problem, because those computational schemes assume a static or quasi-static environment. Furthermore, complete environmental model should be given for the model-based path planner. In order to overcome such limitations, sensor-based reactive control strategy can be adopted [7].

In the sensor-based robot control, the motion of robot is reactively controlled based on sensory information such that obstacles are avoided while the robot continues to move towards the goal in the Obstacle-free direction is chosen based on the sensor. However, it is not guaranteed that the robot reaches its desired goal when only a sensor based reactive control is applied.

A hybrid approach is a combination of the model based planning and a sensor based reactive control. It is possible to achieve advantages of both methods.

Original path toward obstacle-free path. One of the kinds of hybrid approaches adapts path tracking (which is methods aim at keeping the vehicle approximately on a pre-defined path, and bring it back to the path when unacceptable deviations occur. Various approaches for this task have been presented for AGV usage) based on the sensor-based robot control [3].The global dynamic window solved the problem of local minima in the dynamic window approaches. With the real-time global path tracking algorithm, the approaches ensure that the robot is guided to goal position by the local admissible velocities. Despite much progress in the obstacle avoidance researches, the most of researches assumed only the expected dynamic obstacle. To detect the location-dependent unexpected dynamic obstacle, the human cognitive- motivational model that recognizes the afraid of external environment is adapted [4]. Based on the conceptual

model, the quantitative measure of uncertainty and risk is specified. During the autonomous navigation, the robot gathers the data of risky. Thereafter, the robot learns the information of dangerous area from the gathered data. The robot is controlled with the experienced information for the location, kinematics constraints of robot and dynamic information of environment. The experimental result showed that the safe and fast navigation is successfully conducted.

2.2. Requirements for robot path tracking and navigation

Path tracking is the process concerned with how to determine speed and steering settings at each instant of time in order for the robot to follow a certain path. A path consists of a set of points representing the positional coordinates of a particular route. Often when implementing a path tracking algorithm, one also have to implement a path recording unit responsible for saving all the coordinates that constitutes the path. A human operator then has the possibility to manually steer the robot along some track while the path recording unit saves the information about the path. The path tracking algorithm also has to handle unplanned positional or orientation deviations from the path. Such deviations can be caused by odometric errors of some kind, or by new obstacles occurring on the path that must be avoided.

There are many different types of path tracking algorithms available today. Three of them are: follow-the-carrot, pure pursuit, and vector pursuit. The first two methods have been around for quite a while now. The vector pursuit or screw tracking method as it's also called is relatively new. The big difference between these methods is that vector pursuit uses information about orientation at the look-ahead point and others don't.

There are three essential requirements for autonomous mobile robot path tracking and obstacle avoidance Firstly**,** the position and orientation of the robot must be determined. This is known as "localization". Whilst following fixed marking on the floor has been used in industrial settings. This is insufficiently flexible in more general setting odometry can also be used but accumulative errors due the wheel slip, wheel shape distortion or rough terrain soon renders such systems insufficient, on their own, to

provide ongoing reliable localization. The use of artificial beacons is well documented. But the current research goal is to use natural landmarks to support localization.

The second component is sensor based environmental mapping when previously prepared maps have not been provide. Being able simultaneously localizes and map is one of the grand challenges of modern robotics.

The third component is path planning and execution. Most classification path planners aim to devise safest, shortest, and fastest path from a given starting position to a nominated goal. More recently, we have been working in an area we have called "covert robotics" where stealth and un-observability are critical, as they are likely to be in many surveillance activities. Both initially unknown and known with one or more "sentries" and sometimes even a target which must be tracked with minimal exposure can be dealt with in this extension to the classical methodology.

Anther dimension to mobile robotics which has had recent attention is that of cooperative swarms of robots attempting to complete missions more efficiently and robustly than can be achieved with single robots. Tasks such as cleaning, mapping, search and rescue have been posed for such swarms. In the context of "covert robotics" the minimizing of Observability whilst tracking and/or capturing "assailant", through cooperative operation involving multiple robots is of special significance [8].

2.3. Steps of robots navigation

The solving of robot navigation problem can be divided into smaller problems. In order to navigate effectively the robot must:

1. Know where it wants to go
2. Know where it is and what direction it is facing
3. Determine the heading of its destination
4. Steer to and maintain the heading to its destination, and
5. Stop when it has reached its destination.

Knowing where to go: It is the responsibility of higher-level software to specify the course to take. The navigator will only need to provide a means for it to be told where your robot should go next. Therefore, it will need to provide methods that allow higher level software to set the next goal.

The following four methods will serve this purpose:

1. Move To - move to a specified location,
2. Turn To - turn to face a particular direction,
3. Go - move continuously in one direction, and
4. Stop.

Knowing where the robot is: Enabling the robot to keep track of where it is not particularly easy; however, if we have completed studying how to enable the robot to keep track of its position we will know how to create classes that solve this problem. With two shaft encoder sensors and the odometric localizer, the robot will be able to keep track of its position. We can feed position data back into the Navigator by interfacing it to an instance of the Odometric Localizer class [9].

As shown in *Figure 2.1,* the localizers receives the data from the sensors. Using these data the localizer will be able to determine the robot's pose. Knowing the pose will make the navigator decide where to move. This decision can be physically implemented using the servo motors.

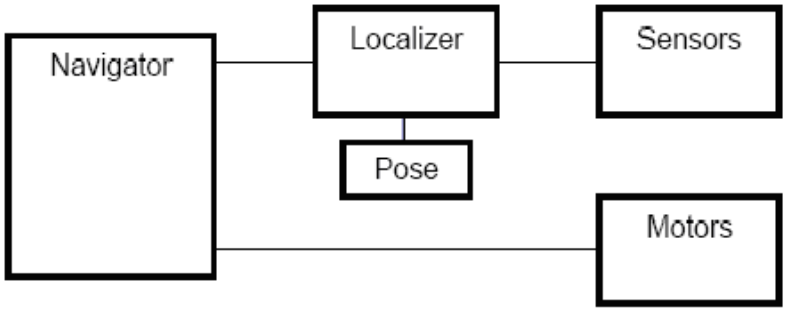


Figure 2.1 Navigation and localization diagram

Determining where to head: Using the output of an Odometric Localizer object, the navigator can use trigonometric relationships to calculate the heading to its next

destination. *Figure 2.2* shows the robot heading toward its destination. The location of the destination relative to your robot's current position in Cartesian coordinates is (xError, yError), the x and y components of the error between where it is and where it is trying to go. The heading is the angle to the destination.

Navigator can calculate xError and yError by subtracting its current x and y. coordinates from those of the destination:

xError = destinationX–x

yError = destinationY–y

The heading is the arc tangent of xError divided by yError:

Heading = arctan(xError / yError)

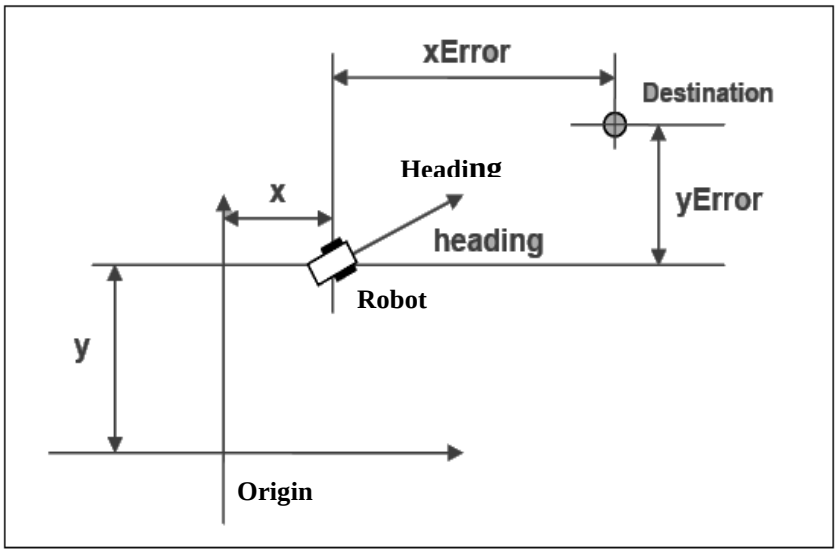


Figure 2.2 Navigating to the destination point

Steering to and maintaining a heading

Once the navigator has calculated the direction the robot needs to head, it must apply power to the motors to steer the robot such that it maintains the heading and moves forward. The key to keeping the robot heading toward its destination is recognizing that it will move approximately straight ahead when the navigator applies the same power to both wheels. The navigator can steer the robot left or right as it moves forward by applying slightly more power to one wheel than the other. The further off course the robot is, the harder your navigator will need to steer to quickly return to the desired

heading; therefore, the larger the error, the larger the power differential it will need to apply to steer back on course. This technique is an application of proportional control, an extremely widely used method for controlling dynamic systems. It gets its name because the output of the controller in this case, the navigator is proportional to the error between the actual value and the target value of the variable being controlled in this case, the robot is heading.

Consider the situation shown in *Figure 2.3.* The robot is heading forward, but slightly off its desired heading. The navigator can correct for the error in heading by applying slightly more power to the right wheel than the left wheel. The amount of power to apply to each wheel can be calculated according to the following equations:

LeftWheelPower = driver Power – differential
RightWheelPower = driver Power + differential

The drive Power is the base level of power applied to the wheels to move straight ahead. The differential variable controls the difference in power applied to the two wheels. The power differential can be made proportional to the heading error using the equation:

Differential = gain * error

Hence, the difference in power applied to the wheels will become larger as the error increases, causing the robot to respond more forcefully to get back on course. When the error is zero both wheels will receive the same power.

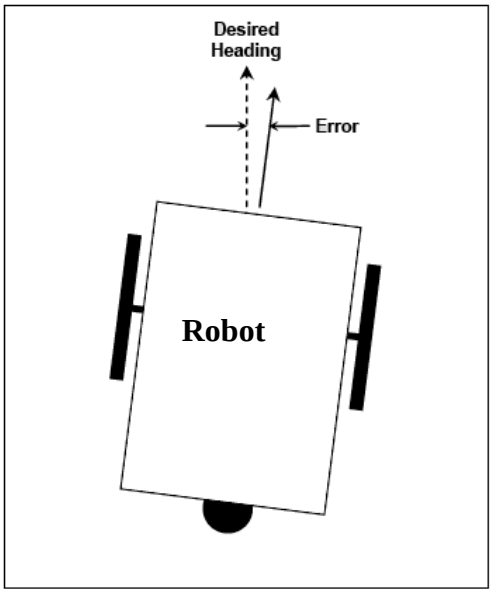


Figure 2.3 Different between desired heading and errors heading

Determining when to stop: The navigator will need to stop the robot when it reaches its destination. The navigator has to turn off the motors when its current position matches the destination. Unfortunately, this process is not simple.

It is easy to forget that the robot is not a high precision system. It will not navigate with perfect accuracy. Although it can navigate to close proximity of its destination, if we insist on extreme precision, we will discover that the robot will get close to its destination and then begin to flounder around attempting to reach the exact destination. In robotics, we usually have to accept close as being good enough. Instead of insisting on perfect navigation, the robot will perform better if we relax our accuracy requirements and program it to stop within reasonable proximity of its desired destination.

The sum of the absolute values of the two error terms, xError and yError can be used, as a rough approximation of how far the robot is from its destination. This will not yield the exact distance, but it will be good enough and far easy to calculate.

2.4. The Boe-Bot robot and its peripheral devices

The robot used in this thesis is called the Boe-Bot robot. In this section, hardware, software, and how to navigate its wheels will be discussed [10].

2.4.1. The Boe-Bot robot hardware

Figure 2.4 below shows a picture of the Boe-Bot robot.

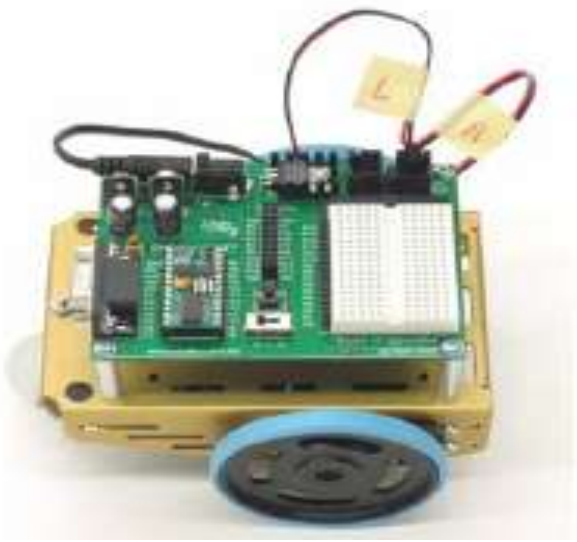


Figure 2.4 The Boe-Bot robot

The major component of the Boe-Bot robot is:

BASIC Stamp 2 module (microcontroller):

The brain of the boe-bot robot is Parallax BASIC Stamp 2 microcontroller (programmable device to sense when a button is pressed) like Avery small computer which plugs into the board of Education carrier board and its removable, can be replace. BASIC Stamp contains (16 I\O pin) for general purpose I\O control. And each I/O pin is protected by a surface mounted 220 Ω resistor.

Basic stamp do four essential robotic tasks:

- Monitor sensors to detect the world around it
- Make decisions based on what it senses
- Control its motion
- Exchange information with its robotics

This chapter shows how to get up and running with BASIC Stamp programming through:

1- Finding and installing the programming software.
2- Connecting BASIC Stamp module to battery power supply.
3- Connecting BASIC Stamp module to the computer for programming.
4- Writing first few BASIC programs.
5- Disconnecting power when we done.

The carrier board makes it easy to connect a power supply and serial cable to the BASIC Stamp module. In later section, and see how the board of education makes it easy to build circuits and connect them to the BASIC Stamp.

The *Figure 2.5* shows the BASIC Stamp module locations on the Boe-Bot board.

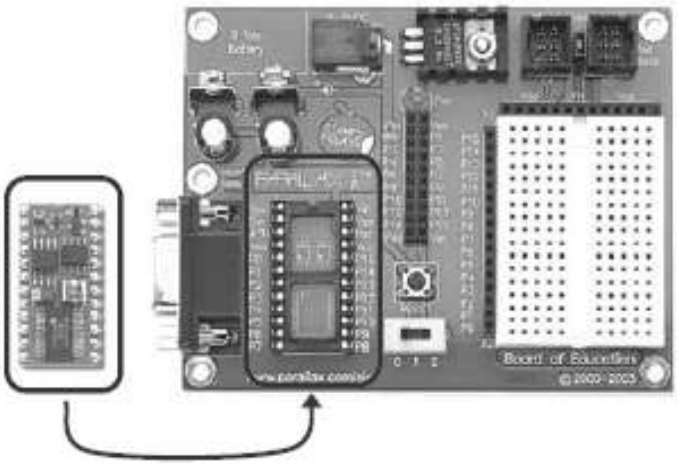


Figure 2.5 BASIC Stamp 2 Modules

Computer Connection Setup:

The board of education or BASIC Stamp should be connected to the PC or laptop with either a serial cable, connect it to available COM port on the back of computer or a USB to Serial Adapter.

Three Position Switch:

The board has 3 position switches. Position-0 is for turning the power to the board completely off. Regardless of whether or not you have battery or power supply connected to the board, when the 3 position switch is set to 0, the device is off. Postion-1 is for power ON / servo ports OFF, Position-2 is for power ON / servo ports ON.

Servo Motors:

The motors that used in Boe-Bot robot are servo motors (Parallax continues rotation servos) the motors that will make the Boe-Bot wheels turn. Standard servos are designed to receive an electronic signal that tells them what position to hold and control the direction, speed, and duration of servo motion. Continuous rotation servos are ideal for controlling wheels.

The signal that has to be sent to the servo connected to pin12 (I\O pin in the microcontroller) to calibrate it this is called the center signal, and after the servo has been properly adjusted, this signal instructs it to stay still.

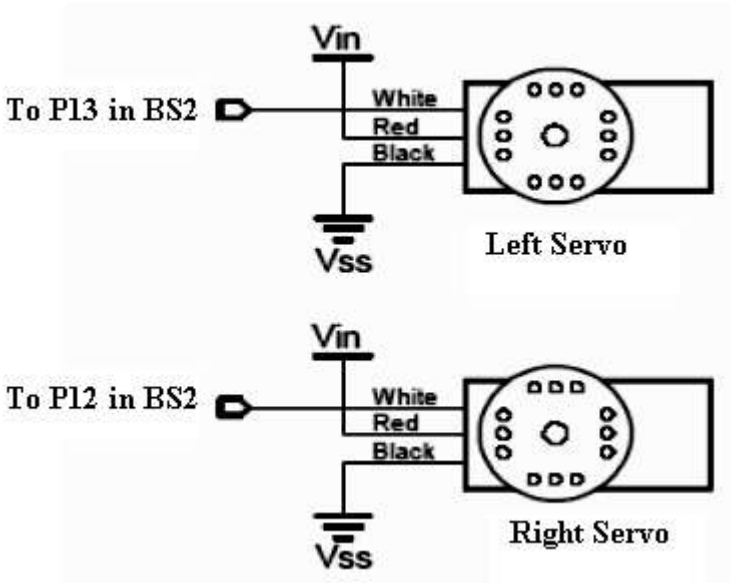


Figure 2.6 Servo connection schematic to the BASIC Stamp

Sending the center signal:

The *Figure 2.7* shows the signal that has to be sent to the servo connected to pin 12. This called the central signal, the instruction consists of a series of 1.5 ms pulses with 20 ms pauses between each pulse.

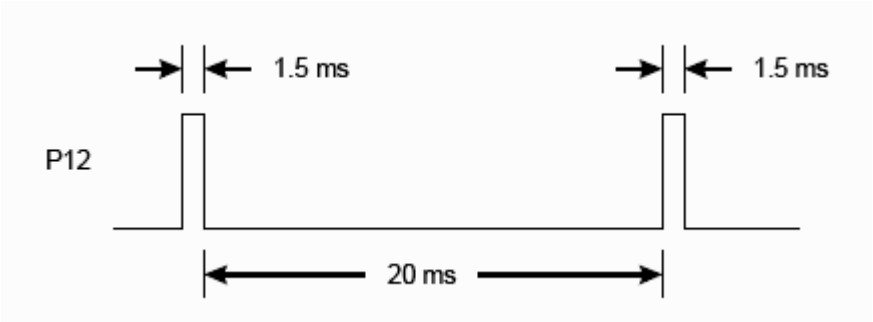


Figure 2.7 Timing diagram for center servo connect to P12

The program for this signal will be a **PULSOUT** command and a **PAUSE** command inside a **DO…LOOP**.

```
DO
      PULSOUT 12, Duration
      PAUSE 20
LOOP
```

Figuring out the PAUSE command will be PAUSE 20 for the 20ms between pulses.

Figuring out the PULSOUT commands pin argument isn't that hard either, its going to be 12, for I/O pin 12. Next, the PULSOUT command's Duration argument has to be for 1.5 ms pulses. 1.5ms is 1.5 thousandths of a second, or 0.0015s. Whatever number is in the PULSEOUT commands Duration argument, multiply that number by 2µs (2 millionths of a second = 0.000002 s), and how long the pulse will last is known.

Also figure out what the PULSOUT commands Duration argument has to be if we know how long we want the pulse to last. Just divided 2µ s into the time we want the pulse to last. With this calculation:

Duration argument = pulse duration / 2µs= 0.0015s/0.000002s=750

We now know that the command for a 1.5ms pulse to P12 will be PULSOUT 12, 750.

It's best to center one servo at a time, because we can hear when the motor stops as it is being adjusted. The system will only send the center signal to the servo connected to P12, and repeat the process with the servo connected to P13 [10].

There's one last thing to do before assembling the Boe-Bot, and that's testing the servos. In this activity, running programs that make the servos turn at different speeds and durations. By doing this, verify that the servos are working properly before you assemble the Boe-Bot.

Pulse Width Controls Speed and Direction:

Recall from centering the servo that a signal with a pulse width of 1.5ms caused the servos to stay still. This was done using a PULSOUT command with Duration of 750.

Now what would happen if the signals pulse width is not 1.5ms?

If BASIC Stamp programmed to send series of pulse that < from 1.5ms pulses for example send series of 1.3ms pulses the Parallax continuous Rotation servo turns full speed clockwise with full speed ranges from 50 to 60 RPM (Revolutions per Minute) as in *Figure 2.8.*

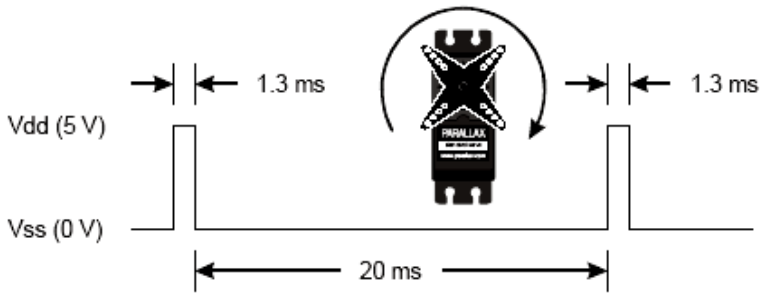


Figure 2.8 A 1.3 ms pulse train turns the servo full speed clockwise

Notice that 1.3ms pulse requires a PULSOUT command Duration argument of 650 which is less than 750. All pulse widths less than 1.5 ms, and therefore PULSOUT Duration argument less than 750, will cause the servo to rotate clockwise.

```
DO
        PULSOUT 12, 650
        PAUSE 20
LOOP
```

If the PULSOUT commands Duration argument greater than 750 will cause the servo to rotate counterclockwise. Duration of 850 will send 1.7ms pulses. This will make the servo turn full speed counterclockwise. As shown in *Figure 2.9.*

```
DO
      PULSOUT 12,850
      PAUSE 20
LOOP
```

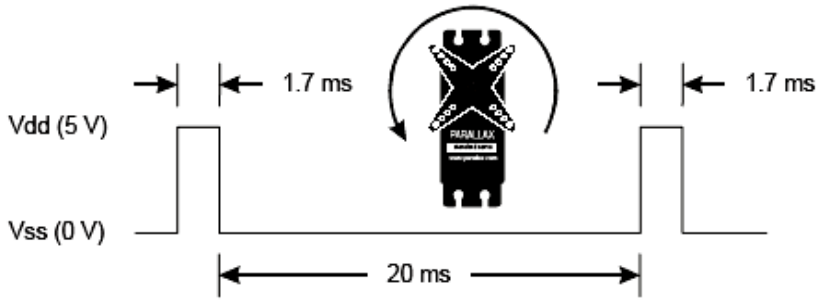


Figure 2.9 A 1.7 ms pulse train makes the servo turn full speed counterclockwise

2.4.2. Boe-Bot robot software

The BASIC Stamp Editor (version 2.0 or higher) is the software used in most of activities and project. This software allows writing programs on the computer and downloads them into Boe-Bot BASIC Stamp brain. It also display messages on the computer screen sent by BASIC Stamp, allowing the Boe-Bot one way to report what it is doing and sensing , the robotics.

It’s easy to download the BASIC Stamp Editor software from the parallax web site. Or from parallax CD.

The BASIC Stamp needs to be connected to power for it to run. It also needs to be connected to a PC so it can be programmed. After making these connections, the BASIC Stamp Editor can be used to test the system [10].

2.4.3 Setting up the hardware and testing the system

Required hardware

The following hardware are needed:

(1) Strip of four rubber feet
(1) Battery pack
(1) BASIC Stamp 2
(1) Board of Education
(4) Batteries

Connecting the hardware

1- Remove each rubber foot from the adhesive strip and affix it to the underside of the board of education.
2- Set the 3- position switch on the board of education to position-0.
3- Load the batteries into battery pack.
4- Insert the BASIC Stamp into the socket on the board.
5- Make sure the pins are lined up properly with the holes in the socket
6- Plug the serial cable into the board.
7- Plug the battery pack into the 6-9 VDC battery jack.
8- Move the 3-position switch from position-0 to position-1 to turn the power on the green light labeled Pwr on the board should now be on.
9- Skip to the testing for communication.

Testing for communication

1- First, run the BASIC Stamp Editor.
2- To make sure that BASIC Stamp is communicating with the computer, click the Run menu, then select Identify.

3- Check the identification window to make sure a BASIC Stamp 2 has been detected on one of COM ports. If it's detected, then system is ready for programming.

2.4.4 Boe-Bot navigation

The Boe-Bot can be programmed to perform a variety of maneuvers: forward, backward, rotate left, rotate right, and pivoting turns. The type of maneuver is determined by the PULSOUT commands Duration arguments. How far the maneuver goes is determined by the FOR….NEXT loops StartValue and EndValue argument [10].

2.4.4.1. Moving forward

To make the Boe-Bot forward, the Boe-Bots wheel has to turn counterclockwise, but its right wheel has to turn clockwise.
An example program for moving the robot forward for three seconds:

```
DEBUG "Program Running!"
Counter VAR   Word
FREQOUT 4, 2000, 3000         , Signal program start/ reset.
FOR counter = 1 TO 122        , Run servos for 3 seconds.
PULSOUT 13, 850
PULSOUT 12, 650
PAUSE 20
NEXT
```

This FOR….NEXT loop sends 122 sets of pulses to the servos, one each to P13 and P12 pausing for 20 ms after each set and then returning to the top of the loop.

PULSOUT 13, 850 cause the left servo to rotate counterclockwise while PULSOUT 12, 650 causes the right servo to rotate clockwise. Therefore, both wheels will be turning toward the front end of the Boe-Bot, causing it to drive forward. It takes about 3

seconds for the FOR…..NEXT loop to execute 122 times, so the Boe-Bot drives forward for about 3 seconds.

The Boe-Bot move forward half the distance. By changing the FOR…NEXT loops ENDValue argument. In the example change from 122 to 61 make Boe-Bot move forward half distance.

And change ENDValue from 122 to 244 make Boe-Bot move forward double of the distance.

The PULSOUT Duration argument of 650 and 850 caused the servo to rotate near their maximum speed. By brining each of the PULSOUT Duration arguments closer to the stay-still value of 750, for slow down the Boe-Bot robot. For example

```
PULSOUT 13, 780
PULSOUT 12, 720
```

2.4.4.2. Moving backward, Rotating, and Pivoting

All it takes to get other motions out of the Boe-Bot are different combinations of the PULSOUT Duration argument. For example, the PULSOUT can be used to make the robot go backwards:

```
PULSOUT 13, 650
PULSOUT 12, 850
```

The commands will make the robot rotate in a left turn (counterclockwise)

```
PULSOUT 13, 650
PULSOUT 12, 650
```

The command will make the Boe-Bot rotate in right turn (clockwise)

```
PULSOUT 13, 850
PULSOUT 12, 850
```

Combine all these commands into a single program that make Boe-Bot move forward, turn left, turn right, and backward.

The Boe-Bot can turn by pivoting around one wheel. The trick is to keep one wheel still while the other rotates.

For example, if the left wheel still and make the right wheel turn clockwise (forward) The Boe-Bot will pivot to the left

```
PULSOUT 13, 750
PULSOUT 12, 650
```

To pivot forward and to the right, simply stop the right wheel, and make the left wheel turn counterclockwise (forward)

```
PULSOUT 13, 850
PULSOUT 12, 750
```

The PULSOUT commands for pivoting backwards and to the right.

```
PULSOUT 13, 650
PULSOUT 12, 750
```

Finally, the PULSOUT commands for pivoting backward and to the left.

```
PULSOUT 13, 750
PULSOUT 12, 850
```

2.4.4.3. Straightening the Boe-Bot's path

The first step is to examine the Boe-Bot's travel for long enough to find out if it's curving (veering) either to the left or to the right when it's supposed to be going straight ahead.

Ten seconds of forward travel should be enough. This can be accomplished with change the EndValue of the For counter from 122 to 407, the command is:

```
FOR counter = 1 TO 407
PULSOUT 13, 850
PULSOUT 12, 650
PAUSE 20
NEXT
```

For example the Boe-Bot turns slightly to the left. There are two ways to think about this problem: either the left wheel is turning too slowly, or the right wheel is turning too quickly. Since the Boe-Bot is already at full speed, speeding up the left wheel isn't

going to be practical, but slowing down the right wheel should help remedy the situation.

As reminded that servo speed is determined by the PULSOUT command's Duration argument. The closer the Duration is to 750, the slower the servo turns. This means change the 650 in the command PULSOUT 12,650 to something a little closer to 750. If the Boe-Bot is only just a little off course, maybe PULSOUT 12,663 will do the trick. If the servos are severely mismatched, maybe it needs to be PULSOUT 12,690. It will probably take several tries to get the right value. Let's say that we first guess is That PULSOUT 12,663 will do the trick, but it turns out not to be enough because the Boe-Bot is still turning slightly to the left. So try PULSOUT 12,670. Maybe that Overcorrects and it turns out that PULSOUT 12,665 gets it exactly right. This is called an iterative process, meaning a process that takes repeated tries and refinements to get to the right value.

If the Boe-Bot curved to the right instead of the left, it means we need to slow down
The left wheel by reducing the Duration of 850 in the PULSOUT 13,850 command.
Again, the closer this value gets to 750, the slower the servo will turn.

2.4.4.4. Tuning the turns

Software adjustments can also be made to get the Boe-Bot to turn to a desired angle, such as 90°. The amount of time the Boe-Bot spends rotating in place determines how far it turns. Because the FOR…NEXT loop controls run time, the FOR…NEXT loop's EndValue argument can be adjust to get very close to the turning angle you want.

```
FOR counter = 1 TO 24       ' Rotate left - about 1/4 turn
PULSOUT 13, 650
PULSOUT 12, 650
PAUSE 20
NEXT
```

If the Boe-Bot turns just a bit more than 90° (1/4 of a full circle). We try to change FOR Counter = 1 TO 23, or maybe even FOR counter = 1 TO 22. If it doesn't turn far enough, increase the run time of the rotation by increasing the FOR…NEXT loops EndValue argument to whatever value it takes to complete the quarter turns.

If the value slightly overshooting 90° and the other slightly undershooting, we try choosing the value that makes it turn a little too far, then slow down the servos slightly. In the case of the rotate left, both PULSOUT Duration arguments should be changed from 650 to something a little closer to 750. As with the straight line Exercise, this will also be an iterative process.

2.4.4.5. Maneuvers- Ramping

Ramping is away to gradually increase or decrease the speed of the servos instead of abruptly starting or stopping. This technique can increase the life expectancy of both robot batteries and the servos.

Programming for ramping:

The key to ramping is to use variables along with constants for the PULSOUT commands Duration argument. For example A FOR….NEXT loops that can ramp the Boe-Bot speed from full stop to full speed ahead. Each time the FOR….NEXT loop repeats itself.

```
PulseCount      VAR      Word
FOR PulseCount =1 TO 100
PULSOUT 13, 750   + PulseCount         '1, 2, 3 …100
PULSOUT 12, 750   – PulseCount
PAUSE 20
NEXT
```

The pulse Count variable increases by 1. PulseCount is 1; it's like using the commands

```
PULSOUT 13, 751
PULSOUT 12, 749
```

The second time through the loop, the value of PulseCount is 2, so it's like using the command

```
PULSOUT 13, 752
PULSOUT 12, 748
```

As the value of the PulseCount variable increases, so does the speed of the servos. By the hundredth time through the loop, the PulseCount variable is 100 so the command is

```
PULSOUT 13, 850
PULSOUT 12, 650
```

This is full-speed ahead for the Boe-Bot.

This used to ramp the speed back down again by using FOR PulseCount = 100 TO 1. Also create routines to combine ramping up or down with the other maneuvers. An example to ramp up to full speed going backward instead of forward. The only difference between this routine and the forward ramping routine is the value of PulseCount is subtracted from 750 in the PULSOUT 13 command, where before it was added. Likewise, PulseCount is added to the value of 750 in the PULSOUT 12 commands, where before it was subtracted.

```
FOR PulseCount = 1 TO 100
PULSOUT 13, 750     -   PulseCount
PULSOUT 12, 750     +   PulseCount
PAUSE 20
NEXT
```

A routine for ramping can be making into a turn by adding the value of PulseCount to 750 in both PULSOUT commands. By subtracting PulseCount from 750 in both PULSOUT commands, also can ramp into a turn the other direction. An example of aquarter turns with ramping. The servos here don't get an opportunity to get up to full speed before they have to slow back down again.

```
'Ramp up right rotates.
```

```
FOR PulseCount = 0   To   30
PULSOUT 13, 750    + PulseCount
PULSOUT 12, 750    + PulseCount
PAUSE 20
NEXT

'Ramp down right rotate
FOR PulseCount = 30   TO   0
PULSEOUT 13,    750      + PulseCount
PULSEOUT 12,    750      +PulseCount
PAUSE 20
NEXT
```

After the Boe-Bot maneuvers description, several possible combinations of PULSOUT Duration arguments are tested. Then studying the behavior of the boe-bot and summaries the result in the *Table 2.1*.

Table 2.1 PULSOUT duration combinations

Duration		**Description**	**Behavior**
P13	**P12**		
850	650	Full speed, p13 counterclockwise, P12 servo clockwise	Forward
650	850	Full Speed P13 CW, P12 CCW	Backward
850	850	Full Speed P13 CCW, P12 CCW	Right rotate
650	650	Full Speed P13 CW, P12 CW	Left rotate
750	850	P13 Stopped P12 CCW Full speed	Pivot back left

750	750	P13 Stopped P12 Stopped	Stopped
760	740	P13 CCW Slow P12 CW Slow	Forward slow
770	730	P13 CCW Med P12 CW Med	Foreword medium
850	700	P13 CCW Full Speed P12 CW Medium	Veer right
800	650	P13 CCW Medium P12 CW Full Speed	Veer left

2.4.4.6. Calculating the travelled distance

Calculate the servo run time by using the speed equation:

Servo run time = Boe – Bot distance / Boe – Bot speed …….. (1)

Which Boe-Bot speed measured in (inch/sec) or (cm/sec).

Example: At 9 in/s, the Boe-Boe has to travel for 2.22 s to travel 20 in.

Time = 20 in / 9 in/s = 2.22 s

And calculate how many pulses to send to the servos. This is the number we will have to use for the FOR……NEXT loop's EndValue argument.

By using this equation:

Pulses = (Boe–Bot distance / Boe – Bot speed) × 40.65 pulse/s…… (2)

Where (40.65pulses/s) is the number of pulses that the basic stamp sends to the servos Each second. This value come from reciprocal of the 24.6 ms (0.024 s) which its each time the two servo PULSOUT and one PAUSE commands are executed in a

FOR…NEXT loop. That it takes 24.6 ms (0.024 s), 1 / 0.024 s/pulse = 40.65 pulses/s.
Example: At 23 cm/s, the Boe-Bot has to travel for 2.22 s to travel 51 cm.
Pulses = (51 cm /23 cm/s) × 40.65 pulses/s = 90 pulses

Now describing the navigation based on sensory inputs, instead of navigation from a pre-programmed list.

2.4.4.7. Boe-Bot navigation using whiskers sensor

In robotics, sensors are used for both internal feedback control and external interaction with the outside environment. There are many different types of sensors available that measure different phenomena. However, in these chapter discussion sensors that is used in conjunction with robotics.

Many type of robotic machinery rely on a variety of tactile switch. The robot can be programmed to pick up the object and place it elsewhere. Factories use tactile switch to count object on a production line, and also for aligning object during industrial processes. In all these instances, the switches provide inputs that dictate some other form of programmed output. The inputs are electronically monitored by the product, be it a robot, or calculator, or a production line. Based on the state switches, the robot arm grabs an object, or the calculator display updates, or the factory production line reacts with motors or servos to guide products.

2.4.4.7.1. Tactile navigation

The whiskers are so named because that is what these bumper switches look like, though some argue they look like antennae. At any rate, this whisker gives the Boe-Bot the ability to sense the world around it through touch, much like the antennae or the whiskers on a cat.

2.4.4.7.2. Whiskers circuit

Before moving on to programs that make the Boe-Bot navigate based on what it can touch its essential to build and test the whiskers first. This activity will guide us through building and testing the whiskers. The *Figure 2.10* shows the building circuit of whiskers and the component that used.

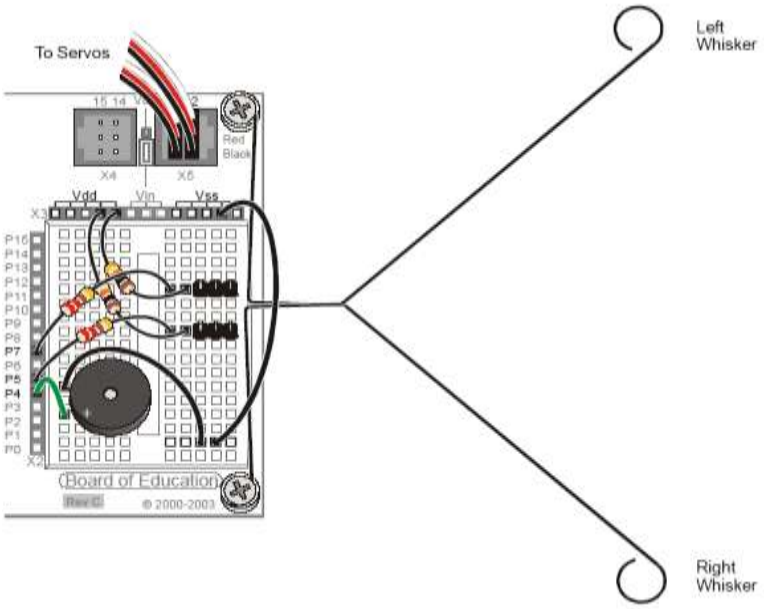


Figure 2.10 Whiskers circuit on the board of education [10]

2.4.4.7.3. Testing the whiskers

Note from schematic *Figure 2.11* that each whisker is both the mechanical extension and the ground electrical connection of normally open, single-pole, single-throw switch. The reason the whiskers are connected to ground (Vss) is because the plated holes at the outer edge of the board are all connected to Vss.

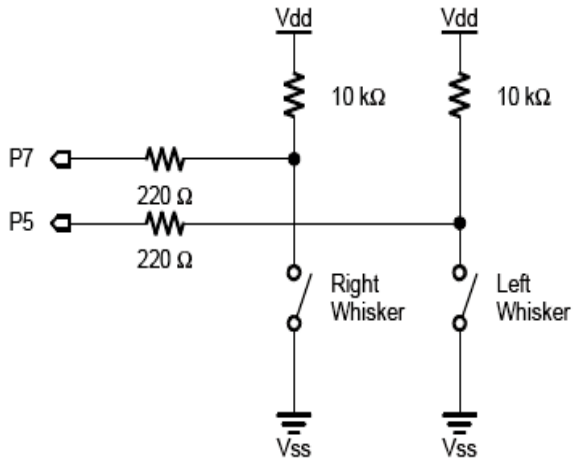


Figure 2.11 Whiskers schematic – a second look

The BASIC Stamp can be programmed to detect when a whisker is pressed. I/O pins connected to each switch circuit monitor the voltage at the 10 kΩ pull-up resister. When a given whisker is not pressed, the voltage at the I/O pin connected to that whisker is 5 V. when a whisker is pressed; the I/O line is shorted to ground (Vss).

The BASIC Stamp can tell us if its ready a 1 or 0 , because the circuit is connected to P7, this 1 or 0 value will appear in variable named INT7 (built-in input register). After program is designed to test the whisker to make sure they are functioning properly. By displaying the binary digits stored in P7 and P5 input registers (IN7 and IN5) the result when press the right whisker into its three-pin header, and note the values displayed in the debug terminal. It should now read:

P5=1 P7=0

The result when press the left whisker into its three-pin header, and note the values displayed in the Debug Terminal. This time it should now read:

P5=0 P7=1

The result when press both whiskers against both three-pin headers.

P7=0 P5=0

2.4.4.7.4. Programming the Boe-Bot to navigate based on whisker input.

The BASIC Stamp was programmed to detect whether a given whisker was pressed. In this, the BASIC Stamp will be programmed to take advantage of this information to guide the Boe-Bot. when the Boe-Bot is rolling along and a whisker is pressed, it means the Boe-Bot bumped into something. A navigation program needs to take this input, decide what it means, and call a set of maneuvers that will make the Boe-Bot back up from the obstacle, turn, and go in a different direction.

Now discuses program make the Boe-Bot go forward until it encounters an obstacle. In this case, the Boe-Bot know when it encounters an obstacle by bumping into it with one or both of its whiskers. As soon as the obstacle is detect by the whiskers, the navigation routines and subroutines will make the Boe-Bot back up and turn. Then, the Boe-Bot resumes forward motion until it bumps into another obstacle. In order to do that, the Boe-Bot needs to be programmed to make decision. PBASIC has a command called an IF….THEN statement that make decisions. The syntax for IF….THEN statement is:

IF (condition) THEN… {ELSEIF (condition)}…. {ELSE}…ENDIF
How the **IF…..THEN** is used

```
IF (IN5 = 0) AND (IN7 = 0) THEN
GOSUB Back_UP         , Both whiskers detect obstacle
GOSUB Turn_Left       , back up &U-turn (left twice)
GOSUB Turn_Left
ELSEIF (IN5 = 0) THEN      , Left whisker contacts
GOSUB Back_UP              , Back up &turn right
GOSUB Turn_Right
ELSEIF (IN7 = 0) THEN      , Right whisker contacts
GOSUB Back_UP              , Back up & turn left
GOSUB Turn_Left
ELSE                       , both whiskers 1, no contacts
GOSUB Forward_Pulse        , Apply a forward pulse &
ENDIF                      , check again
```

The next example program makes decisions based on the whisker inputs, and then calls subroutines to make the Boe-Bot take action.
The program is given in Appendix A (Program 1).

2.4.4.8 Boe-Bot navigation with light sensitive photoresistors

Light has many applications in robotics and industrial control. Some examples include sensing the edge of a roll of fabric in the textile industry, determining when to activate streetlights at different times of the year, when to take a picture, or when to deliver water to a crop of plants.

There are many different light sensors that serve unique functions. The light sensor in the Boe-Bot kit is designed to detect visible light, and it can be used to make the Boe-Bot detect variations in light level. With this ability, the Boe-Bot can be programmed to recognize areas with light or dark perimeters, report the overall brightness and darkness level it sees, and seek out light sources such as flashlight beams and doorways letting light into dark rooms.

It is a light-dependent resistor (LDR) that covers the spectral sensitivity similar to that of the human eye. In other word, the kind of light that your eye detects is the same kind of light that affects the photoresistors resistance. The active elements of these photoresistors are made of Cadmium Sulfide (CdS). Light enters into the semiconductor layer applied to a ceramic substrate and produce free charge carriers. A defined electrical resistance is produced that is inversely proportional to the illumination intensity. In other word, darkness causes more resistance, and brightness cause less resistance. The sensor circuit with photoresistor is shown in *Figure 2.12.*

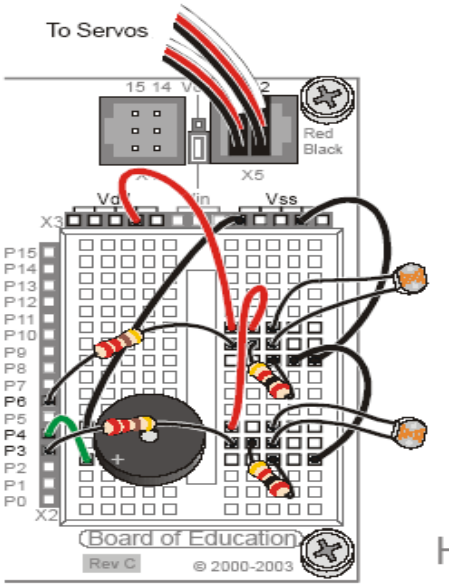


Figure 2.12 Light detection circuit using photoresistor sensors [10]

A BASIC Stamp I/O pin can function as an input or an output. As an output, the I/O pin can send a high (5V) or low (0V) signal. Up to this point, high and low signals have been used to turn LED circuits on and off, control servos, and send tones to a speaker.
A BASIC Stamp I/O pin can also function as an input, the I/O pin dose not apply any voltage to the circuit it is connected to. Instead, it just listens without any actual effect on the circuit.

In this circuit the I/O pin set to input doesn't actually need to have 5 V applied to it to make its input register store a 1. Any thing above 1.4 V (threshold voltage) will make the input register for an I/O pin store a 1. Likewise, an I/O pin doesn't need 0 V to make its input register store a 0. Any voltage below 1.4 V will make an input register for an I/O pin store a 0.

In the *Figure 2.13* shows the equivalent circuit. The resistance of the photoresistoris shown as R changes with light exposure it could be a few Ω if the light is very bright (full light exposure), or it could be in the neighborhood of large resistance for example 50kΩ in complete darkness (shade cast around most of the object).

The voltage at Vo its what the BASIC Stamp I/O pin is detecting when its functioning as an input .its change as photoresistors resistance with light exposure .So the Vo gets larger as R gets smaller, Vo get smaller as R gets larger .

If this circuit connected to IN6, when the voltage is above 1.4 V, IN6 will store a 1. If V_o falls below 1.4 V, IN6 will store a 0.

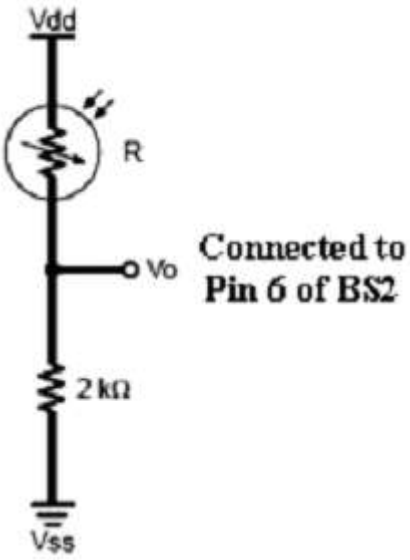


Figure 2.13 Schematic voltage divider circuits [10]

2.4.4.8.1 Detecting shadows

Casting a shadow makes the photoresistors resistance value (R) large, which in turn makes Vo smaller. The 2 KΩ resistors were chosen to make the value of Vo reside slightly above the BASIC Stamp I/O pins 1.4 V threshold in a well lit room. Cast a shadow over it with our hand, it should send Vo below the 1.4 V threshold.

In a well lit room, both IN6 and N3 will store the value 1. A shadow cast over the photoresistor divider connected to P6, it will then store a 0. Likewise, if a shadow cast over the photoresistor divider connected to P3, it will cause IN3 to store a 0.

2.4.4.8.2 Programming photoresistor circuit

Like the programming in the whiskers section adjust the IF…THEN statement so that they monitor IN6 and IN3, instead of IN7 and IN5.

```
Roaming with photoresistor dividers:
IF (IN6 = 0) AND (IN3 = 0) THEN
GOSUB Back_UP
GOSUB Turn_Left
GOSUB Turn_Left
ELSEIF (IN6 = 0) THEN
GOSUB Back_UP
GOSUB Turn_Right
ELSEIF (IN3 = 0) THEN
GOSUB Back_UP
GOSUB Turn_Left
ELSE
GOSUB Forward_Pulse
ENDIF
```

The program is given in Appendix A (Program 2).

2.4.4.9. Boe-Bot navigation with infrared headlights

Today's hottest products seem to have one thing in common: wireless communication. Personal organizers beam data into desktop computer, and wireless remotes let us channel surf. Many remote controls and use signals in the infrared frequency range to communicate, below visible parts; the BASIC Stamp can also receive and transmit infrared light signals.

2.4.4.9.1. Infrared headlight

Infra-red is light (or electromagnetic radiation) that has lower frequency, or longer wavelength than red light. *Table 2.2* shows the wavelengths for common colors along with infrared spectrum. Our IR LED and detector work at 980 nm which is considered near infrared. Night-vision goggles and IR temperature sensing use far infrared wavelengths of 2000-10,000 nm Depending on the application. These sensors react to the intensity of light projected on them by changing their electrical resistance. If the intensity of light, the lower the resistance is and, consequently, higher the current is. As a result, the voltage drop across the switch is less. These sensors are inexpensive and very useful. They can be used for making optical encoders and other devices as well. They are also used in tactile sensors.

Infrared sensors are sensitive to the infrared range. Since infrared is invisible to human eyes, it will not disturb humans if used in devices that project the light out. For example, if a device needs light to measure a large distance for navigation purposes, infrared can be used without attracting attention or disturbing anyone.

Table 2.2 Colors and approximate wavelengths

Color	**Wavelength**	**Color**	**Wavelength**
Violet	400	Red	780
Blue	470	Near infrared	800-1000
Green	565	Infrared	1000-2000
Yellow	590	Far infrared	2000-10,000
Orange	630		

2.4.4.9.2. How infrared headlights works

The infrared object detection system is built on the Boe-Bot is like cars headlights in several respects. When the light from a cars headlights reflects off obstacles, your eye detect the obstacles and your brain processes them and makes your body guide the car Accordingly. The Boe-Bot uses infrared LEDs for headlight as shown in *Figure 2.14.*

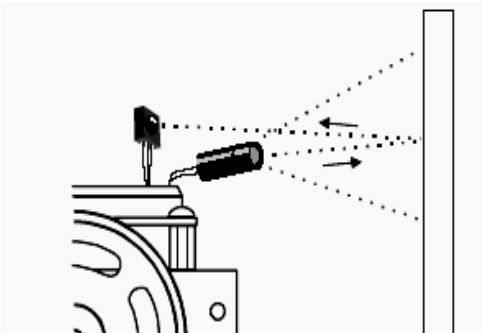


Figure 2.14 Object detection with IR headlights

They emit infrared, and in some cases, the infrared reflects off object and bounces back in the direction of Boe-Bot. the eyes of the Boe-Bot are the infrared detectors. The infrared detectors send signals indicating whether or not they detect infrared reflected off an object. The brain of the Boe-Bot, the Basic Stamp, makes decisions and operates the servo motors based on this sensor input.

The IR detectors have built-in optical filters that allow very little light except the 980 nm infrared that we wanted to detect with its internal photodiode sensor. The infrared detector also has an electronic filter that only allows signals around 38.5 KHz to pass through. In other words, the detector is only looking for infrared that's flashing on and off 38.500 times per second. This prevents IR interference from common sources such as sunlight and indoor lighting. Sunlight is DC interference (0 Hz), and indoor lighting tends to flash on and off at either 100 or 120 Hz, depending on the main power source in the region. Since 120 Hz is outside the electronic filters 38.5 KHz band pass frequency, it is completely ignored by the IR detectors.

2.4.4.9.3. Building the IR circuit

Building and testing the infrared transmitter/detector pairs as shown below in *Figure 2.15* and *Figure 2.16*.

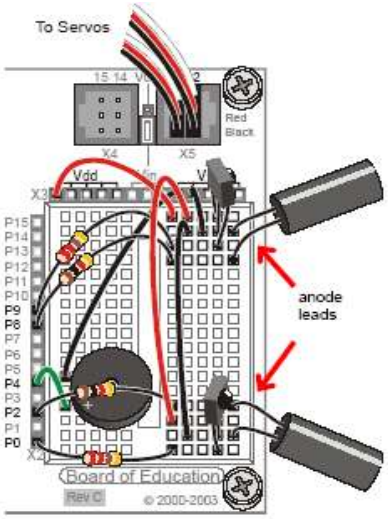


Figure 2.15 Infrared emitter and receive circuit [10]

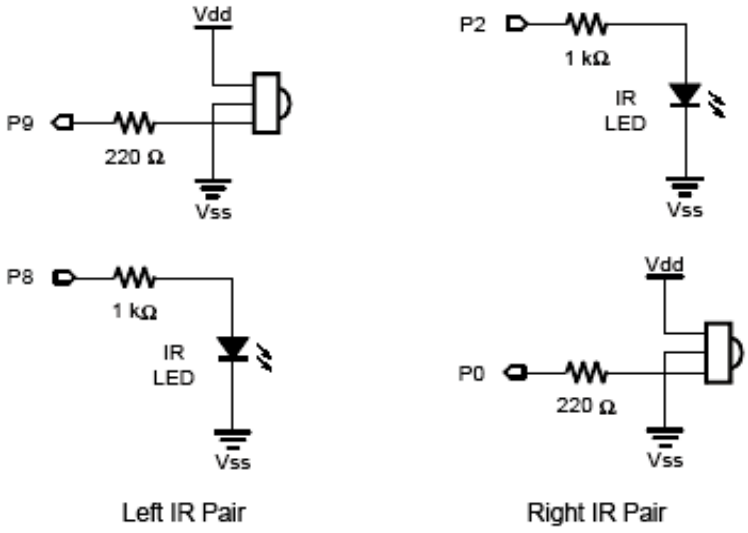


Figure 2.16 Left and right IR LED/detectors pairs [10]

2.4.4.9.4. Testing the IR pairs using the FREQOUT trick

The FREQOUT command was designed mainly to synthesize audio tones. The actual range of the FREQOUT command is 1 to 32768 Hz. One interesting phenomenon of digitally synthesized tones is that they contain signals called harmonics. A harmonic is a

higher frequency a tone is that's mixed in with the tone we want to hear. These tones are outside human abilities to detect sound, which tend to range from 20 Hz to 20 KHz. The harmonics generated by the FREQOUT command start at 32769 Hz and go upward. Control these harmonics directly using Freq1 arguments above 32769 Hz. in this use the command FREQOUT 8, 1, 38500 to send a 38.5 KHz harmonic that lasts 1 ms to P8. The infrared LED circuit connected to p8 will broadcast this harmonic. If the infrared light is reflected back to the Boe-Bot by an object in its bath, the infrared detector will send the BASIC Stamp a signal to let it know that the reflected infrared light was detected.

The key to making each IR LED/detector pair work is to send 1ms of 38.5 KHz FREQOUT harmonic, and then, immediately store the IR detectors output in a variable. Here is an example that sends the 38.5 KHz signal to the IR LED connected to P8, then stores the IR detectors output, which is connected to P9, in a bit variable named irDetectLeft.

```
FREQOUT 8, 1, 38500
IrDetectLeft = IN9
```

The IR detectors output state when it sees no IR signal is high. When the IR detector sees the 38500 Hz harmonic reflected by an object, its output is low. The IR detectors output only stays low for a fraction of a millisecond after the FREQOUT command is done sending the harmonic, so it's essential to store the IR detectors output in a variable immediately after sending the FREQOUT command. The value stored by the variable can then be displayed in the debug terminal or used for navigation decisions by the Boe-Bot.

2.4.4.9.5. Programming for IR object Detection/Avoidance

In programming IR circuit tow bit variables were added to store the states of the Infrared detectors.

```
IrDetectLeft       VAR        Bit
IrDetectRight     VAR         Bit
```

A routine was also added to read the IR pairs.

```
FREQOUT 8, 1, 38500
IrDetectLeft = IN9
```

The IF… THEN statements were modified so that they look at the variables that store the IR pair detection instead of the whiskers inputs.

```
IF (irDetectLeft = 0) AND (irDetectRight = 0) THEN
GOSUB Back_UP
GOSUB Turn_Left
GOSUB Turn_Left
ELSEIF (irDetectLeft = 0) THEN
GOSUB Back_UP
GOSUB Turn_Right
ELSEIF (irDetectRight = 0) THEN
GOSUB Back_UP
GOSUB Turn_Left
ELSE
GOSUB Forward_Pulse
ENDIF
```

This program is given in Appendix A (Program 3).

2.4.4.9.6 Infrared detection range adjustment

Notice that brighter car headlights (or a brighter flashlight) can be used to see objects that are further away when it's dark. By making the Boe-Bot's infrared LED headlights brighter, also increase its detection range. By resisting electric current less, a smaller resistor allows more current to flow through an LED. More current through an LED is what causes it to glow more brightly. So the effect of different resistance values on infrared LEDs was examined.

In the last circuit, if different value of resistance is connected in series to the IR LEDs instead of the 1 KΩ in the last *Figure2.16*. For example, if the values of these resistors were 220 Ω, 470 Ω, 2 KΩ, and 4.7 KΩ, the result is that resistances which are less than 1 KΩ will make an LED glow more brightly (220 Ω, 470 Ω) and vise versa about 2 KΩ and 4.7 KΩ.

Reasonable hypothesis would be that brighter IR LEDs can make it possible to detect Objects that are further away.

2.5. Summary

Obstacle avoidance and path tracking of Mobile Robots is a broad topic, covering a large spectrum of different technologies and applications. Robotic obstacle avoidance involves three distinct aspects (global navigation, local navigation, personal navigation).

There are three essential requirements for autonomous mobile robot obstacle avoidance and path tracking. Firstly called "localization". The second component is sensor based environmental mapping. The third component is path planning and execution.

This chapter introduced the basic Boe-Bot maneuvers: forward, backward, rotating in Place to turn to the right or left, and pivoting. The type of maneuver is determined by the PULSOUT commands' Duration arguments. How far the maneuver goes is determined by the FOR…NEXT loop's StartValue and EndValue arguments.

Ramping was introduced as a way to gradually accelerate and decelerate. Ramping is accomplished by taking the same variable that's used as the counter argument in a FOR…NEXT loop and adding it to or subtracting it from 750 in the PULSOUT command's Duration argument.

In this chapter the Boe-Bot was programmed to navigate based on sensory inputs. The sensory inputs used in this chapter Firstly the whiskers, which served as normally open contact switches. When properly wired, these switches can show one voltage (5 V) at the switch's contact point when it's open and a different voltage (0 V) when it's closed.

The BASIC Stamp I/O pin's input registers store "1" if they detect Vdd (5 V) and "0," if they detect Vss (0 V). The second sensory inputs is photoresistors pair was used to measure differences in visible light , and the BASIC Stamp monitored the Voltage at the connection between the photoresistor and the fixed resistor. When this Voltage dropped below or raised above 1.4 V the input register for the I/O pin connected to the circuit stored either a 0 or 1. The third sensory input is infrared headlight infrared object detection that uses the Infrared LED found in common handheld remotes, and other appliances that are controlled by these remotes. By Shining infrared into the Boe-Bot's path and looking for its reflection.

CHAPTER THREE. ALGORITHMS FOR OBSTACLE AVOIDANCE AND PATH TRACKING

3.1. Overview

Distance is just one kind of value that robots and other automated machinery are responsible for. When a machine is designed to automatically maintain a value, such as distance, pressure, or fluid level, it generally involves a control system. These systems sometimes consist of sensors and valves, or sensors and motors, or, in the case of the Boe-Bot, sensors and continuous rotation servos. There is also some kind of processor that takes the sensor measurements and converts them to mechanical action. The processor has to be programmed to make decisions based on the sensor inputs, and then control the mechanical outputs accordingly. In the case of the Boe-Bot, the processor is The BASIC Stamp (2).

The sensors (infrared, ultrasound) were used to detect whether an object is in the Boe-Bot's way without actually touching it. To find distance from obstacle it sends a pulse of sound out and records how long it takes for the echo to come back. The time it takes for the echo to come back can then be used to calculate how far away the object is. There is, however, a way to accomplish distance detection with the very simple circuit. With the Boe-Bot able to determine the distance of an object, it can be programmed to follow a moving object without colliding into it. The Boe-Bot can also be programmed to follow any track.

3.2. Measurement of information through sensors

3.2.1. Determining distance with the IR LED/DETECTOR circuit

The infrared circuit was built to detect distance using frequency sweep technique. Set the station for one frequency, and check the output.

In this work frequency sweep is technique of testing a circuit output using a variety of input frequencies.

Figure 3.1 shows an example of how the Boe-Bot can test for distance using frequency. In this example, the object is in Zone 2. That means that the object can be detected when 37500, 38250 and 39500 Hz is transmitted, but it cannot be detected with 40500, and 41500 Hz. If you were to move the object into Zone 1, then the object can be detected when 37500, 38250, 39500, and 40500 Hz are transmitted, but not when 41500 Hz are transmitted. And so on.

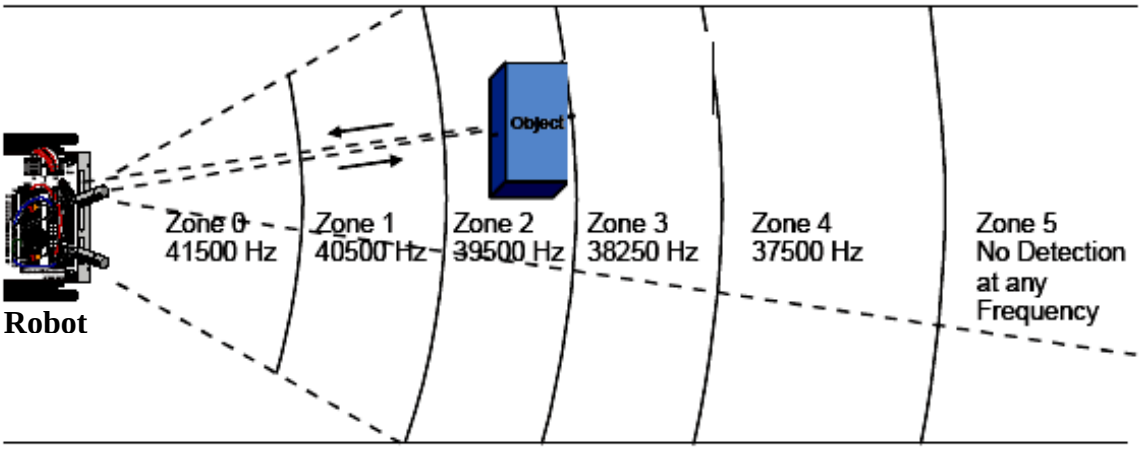


Figure 3.1 Frequencies and zones for the Boe-Bot [20]

The graph in *Figure 3.2* below shows how much less sensitive this IR detector becomes if the IR signals it receives flashes on/off at a frequency other than 38.5 kHz. For example, if you send it IR flashed on/off at 40 kHz, it's only 50% as sensitive as it would be at 38.5 kHz. If the IR is flashed on/off at 42 kHz, the detector is only 20% as sensitive. Especially for frequencies that make the detector less sensitive, the object has to be closer to make the reflected IR brighter for the detector to detect it.

Another way to makes distance detection more simple that the most sensitive frequency will detect the objects that are the farthest away, while less sensitive frequencies can only be used to detect closer objects. So testing the 5 frequencies from most sensitive to least sensitive and depending on which frequency makes the reflected infrared no longer visible to the IR detector, you can infer the distance.

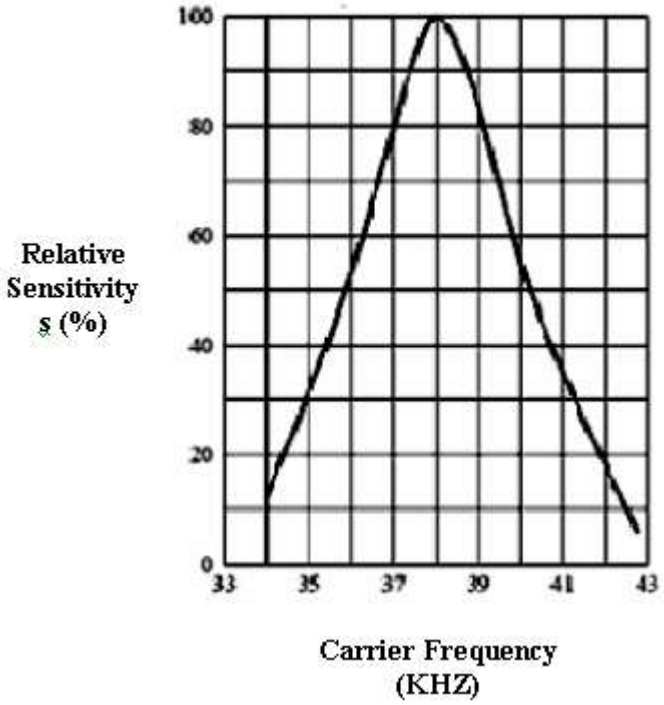


Figure 3.2 Filter sensitivity depends on carrier frequency [20]

In *Figure 3.3* compares the left axis of the graph (IR frequency) to the relative sensitivity of the IR detector. The right side of the graph shows how the relative sensitivity of the IR detector relates to distance detection. As detector sensitivity decreases with the increase in frequency, the object must be closer for the IR signal to be detected. Why closer? When the detectors are made less sensitive by sending higher frequencies, it's like giving them darker and darker lenses to look through. Just as a flashlight beam appears brighter when reflected off an object that's closer to you, IR reflected off a closer object appears brighter to the IR detectors.

The right axis of *Figure 3.3* shows how different frequencies can be used to indicate in which zone a detected object is located. By starting with a frequency of 38.5 kHz, Whether or not an object is in Zone 1-5 can be determined. If an object is not yet detected, it must be beyond the detector limit (Zone 0). If an object is detected, by testing again at 39.25 kHz, the first datum about distance is collected. If 38.5 kHz is detected the object but 39.25 kHz did not, the object must be in Zone 1. If the object was detected at both frequencies, but not at 40.5 kHz, we know it's in Zone 2. If all

three frequencies detected the object, but it was not detected at 41.75 kHz, we know it is in Zone 3. If all four frequencies detected the object, but not 42.5 kHz, we know it's in Zone 4. If all the frequencies detected the object, we know it's in Zone 5.

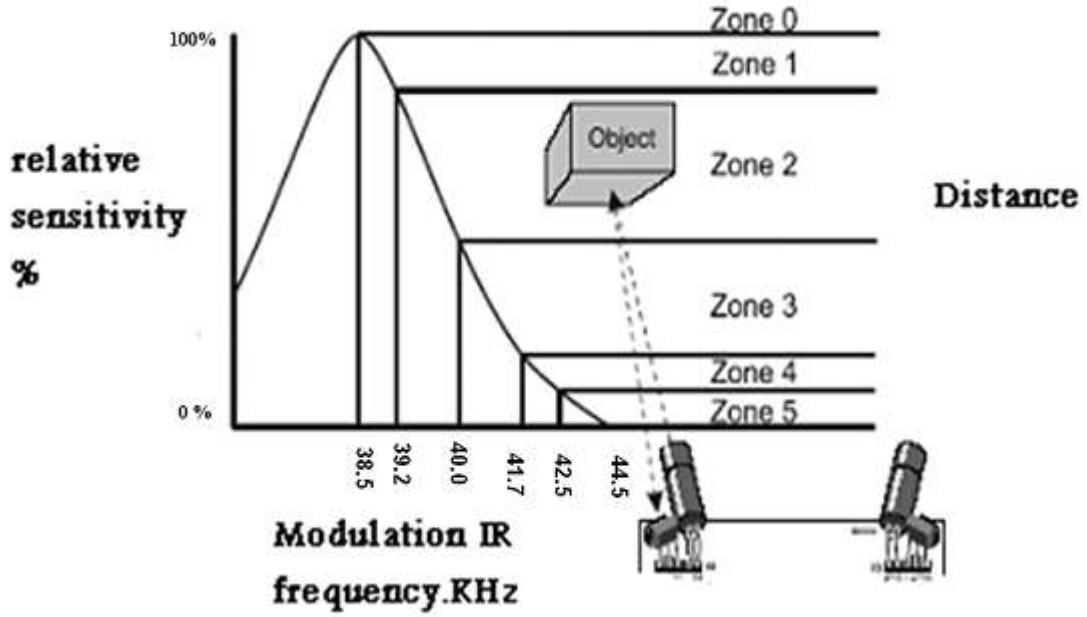


Figure 3.3 Relative IR sensitivity to frequency [20]

The Programming Frequency Sweep for distance detection start with test the IR detector at each frequency, need the FREQOUT to send five different frequencies and test at each frequency to find out whether the IR detector could see the object. The steps between each frequency are not quite even enough to use the FOR…NEXT loop's STEP operator. Five different FREQOUT commands could use, but that would be a waste of code space. Instead, the best approach for storing a short list of values that used in sequence is a command called LOOKUP. The syntax for the LOOKUP command is:

```
LOOKUP Index, [Value0, Value1 … ValueN], Variable
```

If the Index argument is 0, Value0 from the list inside the square braces will be placed in variable. If Index is 1, Value1 from the list will be placed in Variable. There could be up to 256 values in the list, but for our example program, only need 5 here is how it will be used:

```
FOR freqSelect = 0 TO 4
LOOKUP freqSelect, [37500, 38250, 39500, 40500, 41500],
irFrequency
FREQOUT 8, 1, irFrequency
irDetect = IN9
NEXT
```

The first time through the FOR…NEXT loop, freqSelect is 0, so the LOOKUP command places the value 37500 in the irFrequency variable. Since irFrequency contains 37500 after the LOOKUP command, the FREQOUT command sends that frequency to the IR LED connected to P8. And the value of IN9 is then saved in the irDetect variable.

The second time through the FOR…NEXT loop, the value of freqSelect is now 1, which means the LOOKUP command places 38250 into the irFrequency variable, and the process is repeated for this higher frequency. The third time through, it's repeated again with 39500, and so on.

Example: program to test left frequency sweep

TestLeftFrequencySweep.bs2 does two things. First, it tests the left IR LED/detector pair (Connected to P8 and P9) to make sure they are functioning properly for distance Detection. It also demonstrates how the frequency sweep is accomplished. After running the program, the debug terminal will display the zone measurement shown in *Figure 3.4* there are many possible yes-no patterns that can be generate. The test patterns will vary depending on the characteristics of the filter inside the IR detector. The program determines which zone the detected object is in by counting the number of "No" occurrences. Notice that in Figure 3.4 that three "Yes" and two "No" occurrences. Therefore, "Zone 2" is the location of the object detected in this example.

The program is given in Appendix A (Program 4).

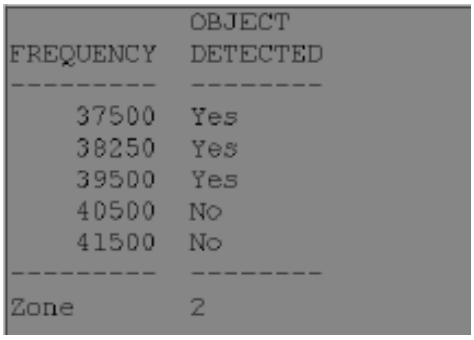


Figure 3.4 An example of testing distance detection

3.2.2. Measurement the responsively shadow controlled Boe-Bot using photoresistor sensor

The Boe-Bot can be more responsive. This wasn't really possible with the whiskers, because the Boe-Bot had to back up before turning since it had already made physical contact with the obstacle. When shadows are used to guide the Boe-Bot, it can check between each pulse to see if the shadow is still detected regardless of whether it's moving forward or executing a maneuver.

One interesting form of remote control is to have the Boe-Bot sit still in normal light, then follow a shadow that cast over the photoresistors. It's kind of a user-friendly way of guiding the Boe-Bot's motion. In the path tracking test and calibrate the Boe-Bot's light sensors so that they recognize the difference between ambient light and a directed flashlight beam.

The Boe-Bot can be programmed to stay still when no shadow is cast over its photoresistor dividers. And when the shadow cast over both photoresistors, the Boe-Bot should move forward. If cast a shadow over one of the photoresistors, the Boe-Bot should turn in the direction of the photoresistor that senses the shadow.

3.2.2.1. Building circuit

Figure 3.5 shows the built circuit that is used for path tracking of the Boe-Bot.

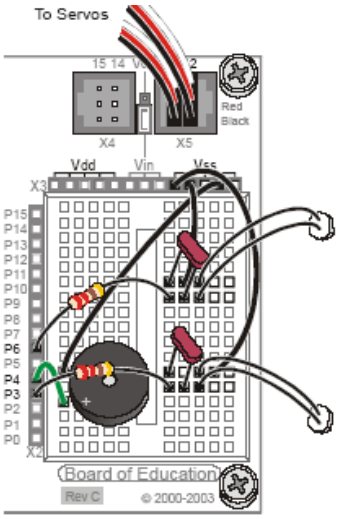


Figure 3.5 Photoresistor sensors orientation [10]

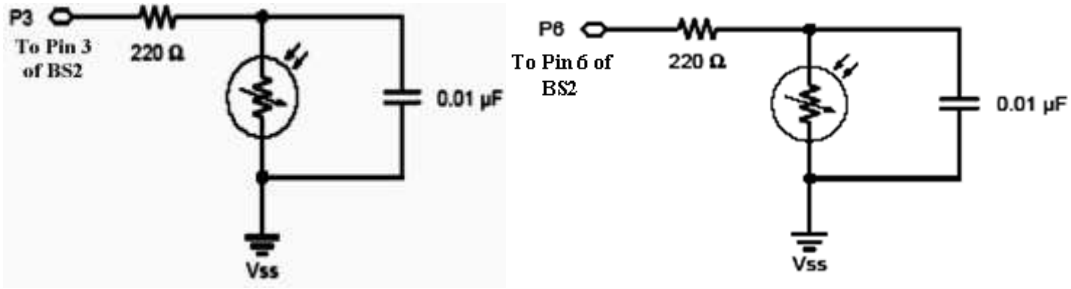


Figure 3.6 Schematic two photoresistor RC circuits [10]

3.2.2.2. RC decay time and the photoresistor circuit

Think of a capacitor in the circuit shown in *Figure 3.7* as a tiny rechargeable battery. When P6 sends the high signal, it essentially charges this capacitor-battery by applying 5V to it. After a few ms, the capacitor charges up to almost 5 V. If the BASIC Stamp Program then changes the I/O pin so that it just quietly listens, the capacitor loses its Charge through the photoresistor. As the capacitor looses its charge through the photoresistor, its voltage decays, getting lower and lower as it looses charge. The

amount of time it takes for the voltage that IN6 senses to drop below 1.4 V depends on how strongly the photoresistor "resists" the flow of electric current supplied by the capacitor. If the photoresistor has a large resistance value due to very dim lighting conditions, the capacitor takes longer to discharge. If the photoresistor has a small resistance value because the light incident on its surface is very bright, it will not resist current very strongly, and the capacitor will lose its charge very rapidly.

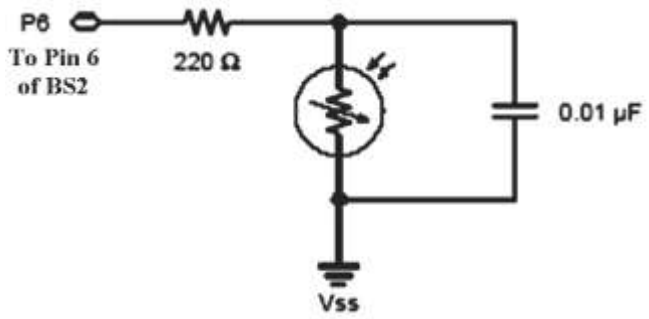


Figure 3.7 RC circuit connected to pin 6 of the microcontroller [10]

3.2.2.3. Measuring RC decay time with the BASIC Stamp

The BASIC Stamp can be programmed to charge the capacitor and then measure the time it takes the capacitor's voltage to decay to 1.4 V. This decay time measurement can be used to indicate the photo resistor's resistance. This resistance in turn indicates how bright the light detected by the photo resistor really is. This measurement requires a Combination of the HIGH and PAUSE commands along with a new command called RCTIME. The RCTIME command is designed to measure RC decay time on a circuit like the one in *Figure 3.7*. Here is the syntax for the RCTIME command:

```
RCTIME Pin, State, Duration
```

The Pin argument is the number of the I/O pin that you want to measure. For example, if P6 measure, the Pin argument should be 6. The state argument can either be 1 or 0. It should be 1 if the voltage across the capacitor starts above 1.4 V and decays downward. It should be 0 if the voltage across the capacitor starts below 1.4 V and grows upward. For the circuit in *Figure 3.7*, the voltage across the capacitor will start close to 5 V and decay to 1.4 V, so the State argument should be 1. The Duration argument has to be a variable that stores the time measurement, which is in µs units.

For example:
Program measure the RC decay time on the photoresistor circuit connected to P6, which is the photoresistor on the Boe-Bot's left.
To get the measurement, the code implements these three steps:
1. Start charging the capacitor by connecting the circuit to 5 V (using the HIGH command).
2. Use PAUSE to give the HIGH command enough time to charge the capacitor in the RC circuit.
3. Execute the RCTIME command, which sets the I/O pin to input, measures the decay time (from almost 5 V to 1.4 V), and stores it in the timeLeft variable.
This program is given in Appendix A (Program 5).

3.2.3. Closed loop control

For one Boe-Bot to follow another, the Boe-Bot that follows, is the shadow vehicle, and has to know how far ahead the lead vehicle is. If the shadow vehicle is lagging behind, it has to detect this and speed up. If the shadow vehicle is too close to the lead vehicle, it has to detect this as well and slow down. If it's the right distance, it can wait until the measurements indicate it's too far or too close again.

Closed loop control is a common method of maintaining levels, and it works very well for helping the Boe-Bot maintain its distance from an object. There are lots of different kinds of closed loop control. Some of the most common are hysteresis, proportional, integral, and derivative control.

Most control techniques can be implemented with just a few lines of code in PBASIC. In fact, the majority of the proportional control loop shown in *Figure 3.8* below reduces to just one line of PBASIC code. This block diagram, describes the steps of the proportional control process that the Boe-Bot will use to measure distance with its right IR LED and detector and adjust position to maintain distance with its right servo.

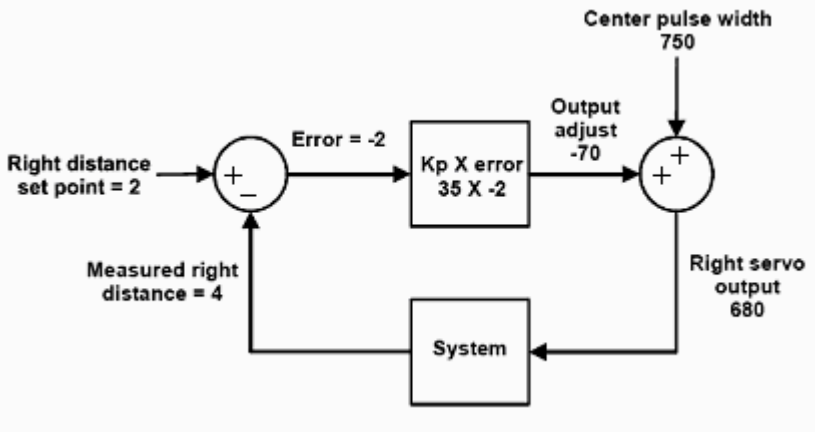


Figure 3.8 proportional control block diagram for right servo and IR LED and detector pairs

This particular example is for the right IR LED/detector and right servo.

-The set Point is 2, which means we want the Boe-Bot to maintain a distance of 2 between itself and any object it detects.

-The measured distance is 4, which is too far away.

-The error is the set point minus the measured distance which is 2 – 4 = –2.

-Next, the error feeds into an operator block. This block shows that error will be multiplied by a value called a proportional constant (Kp). The value of Kp is 35.

-The block's output shows the result of –2 × 35 = –70, which is called the output adjust. This output adjusts goes into another summing junction, and this time it is added to the servo's center pulse width of 750.

The result is a 680 pulse width that will make the servo turn about ¾ speeds clockwise. That makes the Boe-Bot's right wheel roll forward, toward the object. This correction goes into the overall system, which consists of the Boe-Bot, and the object, that was at a measured distance of 4. The equation that can be taken from this block diagram:

.

Error = Right distance set point – Measured right distance
= 2 – 4

Output adjust = error × Kp
= –2 × 35 = – 70

Right servo output = Output adjust + Center pulse width
= – 70 + 750 = 680

The three equations above can be reduced to this one, which will give you the same result.

Right servo output = (Right distance set point – Measured right distance) × □ Kp + Center pulse width

= ((2 – 4) × □ 35) + 750= 680

For The left servo and IR pair has a similar algorithm shown in *Figure 3.9* below the difference is that Kp is –35 instead of +35. Assuming the same measured value at the right IR pair, the output adjust results is a pulse width of 820. Here is the equation and calculations for this block diagram:

Left servo output = (Left distance set point – Measured left distance) × Kp + Center pulse width

= ((2 – 4) × –35) + 750 = 820

The result of this control loop is a pulse width that makes the left servo turn about ¾ of Full speed counterclockwise. This is also a forward pulse for the left wheel. The idea of Feedback is that the system's output is re-sampled.

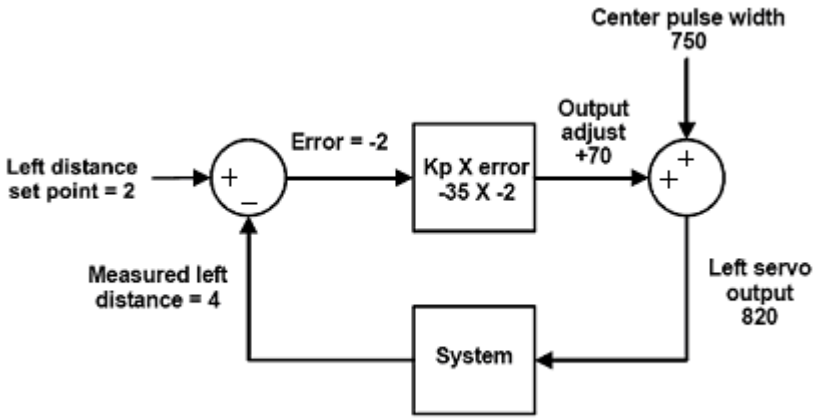


Figure 3.9 Proportional control block diagram for left servo and IR LED and detector pairs

Programming the Boe-Bot example:

In this example of solving the same last equation in PBASIC. The right distance set Point is 2, the measured distance is a variable named distanceRight that will store the IR distance measurement, Kp is 35, and the center pulse width is 750:

pulseRight = 2 - distanceRight × (35) + 750

The left servo is different because Kp for that system is -35

pulseLeft = 2 - distanceLeft × (-35) + 750

Since the values -35, 35, 2, and 750 all have names, it's definitely a good place for some Constant declarations.

```
Kpl          CON    -35
Kpr          CON     35
SetPoint     CON     2
CenterPulse  CON    750
```

With these constant declarations in the program, can use the name Kpl in place of –35, Kpr in place of 35, SetPoint in place of 2, and CenterPulse in place of 750. After These constant declarations, the proportional control calculations now look like this:

```
pulseLeft = SetPoint - distanceLeft × Kpl + CenterPulse
pulseRight = SetPoint - distanceRight × Kpr + CenterPulse
```

The convenient thing about declaring constants for these values is that change them in one place, at the beginning of the program. The changes you make at the Beginning of the program will be reflected everywhere these constants are used. For Example, by changing the Kpl CON directive from -35 to -40, every instance of Kpl in the entire program changes from -35 to -40. This is exceedingly useful for experimenting with and tuning the right and left proportional control loops.

Following BoeBot programming example repeats the proportional control loop just discussed with every Servo pulse. In other words, before each pulse, the distance is measured and the error Signal is determined. Then the error is multiplied by Kp, and the resulting value is Added/subtracted to/from the pulse widths that are sent to the left/right servos.

This program is given in the Appendix A (Program 6).

3.2.4. Following a stripe

In this activity the infrared detectors looking down at the table surface as shown in the *Figure 3.10* below. The program should make it continue forward so long as both IR detectors can see the surface of the table. In other words. The boe-bot can continue forward so long as the table top it's navigating on is detected.

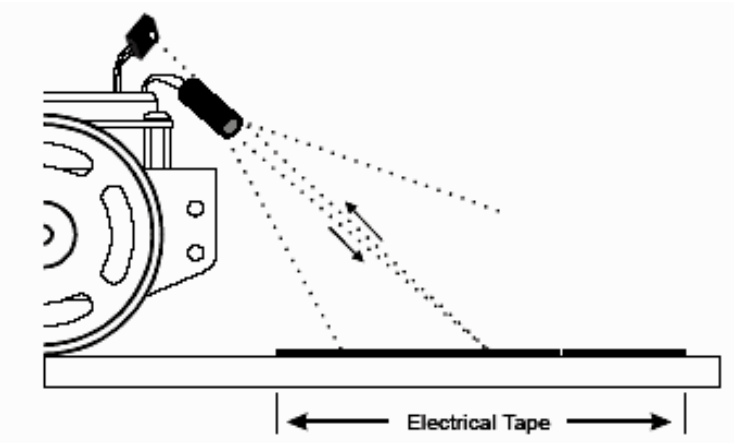


Figure 3.10 IR pairs directed downwards to scan for a drop-off

Building and testing the course:
In this activity we need A sheet of white poster board with a border made of electrical tape (type of pressure sensitive tape used to insulate electrical wire and other material that conduct electricity and it made of plastic) makes for a handy way to simulate the drop-off presented by a table edge, with much less risk to the Boe-Bot.

In this activity the circuit which we used the same infrared LED circuit in the chapter 1 with the drop-off detector and with Replacing the 1 kΩ resistors with 2 kΩ resistors to connect P2 to Its IR LED and P8 to its IR LED. Because the Boe-Bot need to be nearsighted for this activity .and makes sure that the electrical tape course is free of fluorescent light interference. As in *Figure 3.11.*

In *Figure 3.12* when the Boe-Bot is placed on the course so that its wheels straddle the black line the IR detectors should be facing slightly outward. Verify that the distance reading for both IR pairs is 0 or 1 again. If the readings are higher, it means they need to

be pointed slightly further outward, away from the edge of the stripe. When the Boe-Bot is moved in either direction indicated by the double-arrow, one or the other IR pair will become focused on the electrical tape. In this case the readings for the pair that is now over the electrical tape should increase to 4 or 5. Keep in mind that if the Boe-Bot is moved toward its left, the right detectors should increase in value, and if the Boe-Bot moves toward its right, the left detectors should show the higher value.

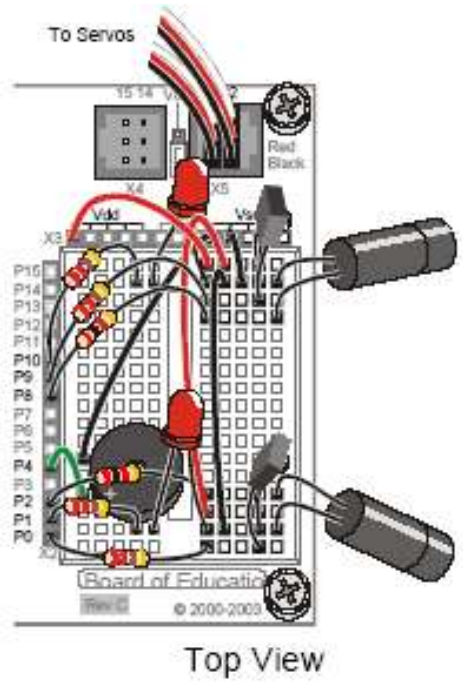


Figure 3.11 The drop-off IR LEDs detectors pair's circuit [10]

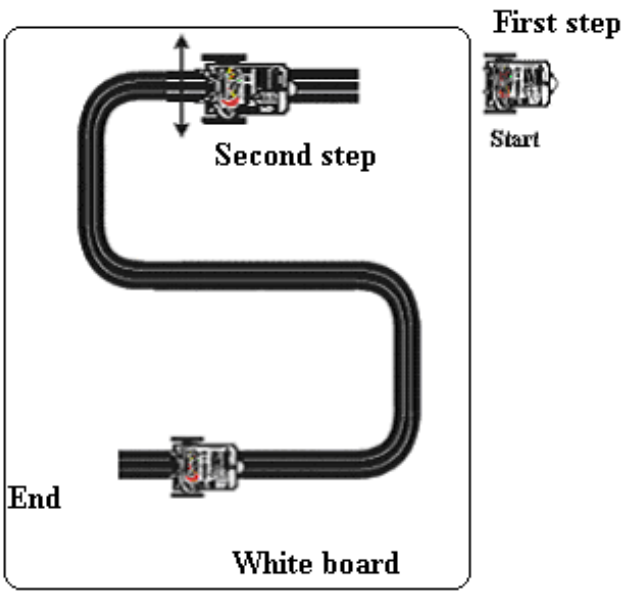


Figure 3.12 Top view of the Boe-Bot delimited strip test

Programming for stripe following:

Need a few small adjustments to following Boe-Bot programming from last to make it work for following a stripe. First, the Boe-Bot should move toward objects closer than the SetPoint and away from objects further from the SetPoint. This is the opposite of how FollowingBoeBot.bs2 behaved. To reverse the direction the Boe-Bot moves when it senses that the object is not at the SetPoint distance, simply change the signs of Kpl and Kpr. In other words, change Kpl from -35 to 35, and change Kpr from 35 to -35. You will need to experiment with your SetPoint. Values from 2 to 4 tend to work best. This next example program will use a SetPoint of 3.

This program is given in Appendix A (Program 7).

3.3. Algorithms for obstacle avoidance and path tracking

3.3.1. Algorithm of calculating distance and angle for goal navigation

In robotic domain, there are various classes of problems; navigation, behavior generation, recognition of the environment, localization. In these problems, localization

is especially indispensable technique for robot system. The problem of estimating the position of mobile robot is one of the fundamental problems in mobile robots filed in this chapter proposed algorithms which search in the most suitable goal position with obstacle avoidance, and path tracking.

The problem involves navigating a robot safely reaches to the goal. This is difficult as the only information available is the position of the robot and the position of the goal. To solve the problem, devised the algorithms. That were developed to solve the problem is able to take care of obstacles without any prior knowledge of their location and safely clear them. It also allows the robot to reach the goal in a shortest time. So that the algorithms satisfied the requirement that the robot has to reach the goal. And follow any desired track.

The sensor which used in this navigation is ultrasound sensor instead of infrared sensor because the infrared that we have measure for short distance. And we need sensor to detect longer distance possible.

3.3.1.1. Methodology

The procedure of our method of estimating the goal position consist the following steps:

1- Estimate the current position of the Boe-Bot robot and the goal, supposed that the robot position is (0, 0) coordinate and in the direction of the x axis in the first quarter of the 2-D plane as the *Figure 3.13* shown.
2- Determining the angle which the robot must turn to stand on the beginning of the correct goal road.
3- Detecting the obstacle's location using ultrasound sensor to measure the shortest distance to the goal.

Limitations: the dimension of the obstacles in this algorithm is limited with maximum length = 40 cm, and maximum width = 15 cm.

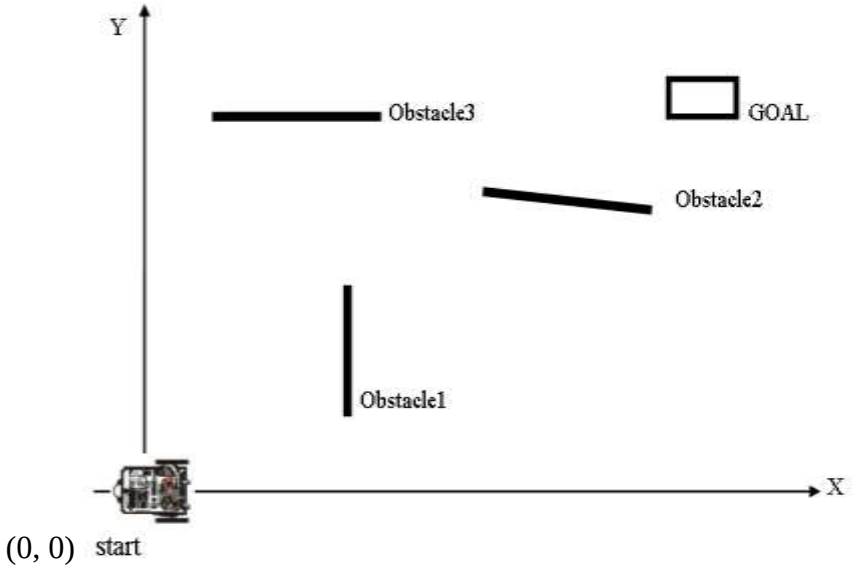


Figure 3.13 Model of the Boe-Bot robot, obstacles, and goal were located in the first quarter of the 2D- plan

3.3.1.2 Determining the angle

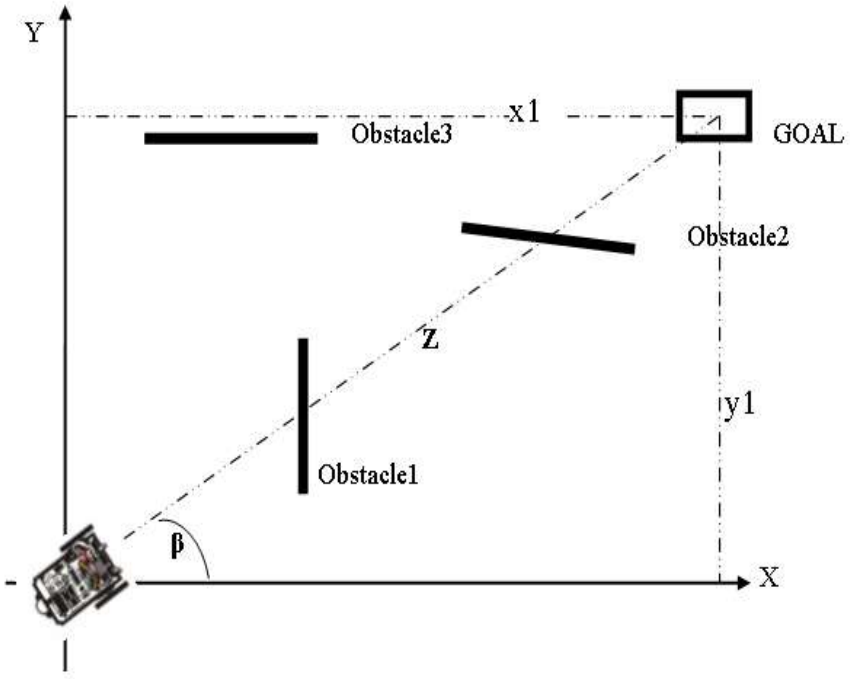


Figure 3.14 Goal and the robot coordinate are used in calculate β and Z

From the *Figure 3.14* calculate the distance (z) by using equation 1:

$Z = [(x1)^2 + (y1)^2]$…… (1)

Where (x1, y1) is the goal coordinating.

And the Z is the straight line between the robot and the goal.

By using 2 and 3 equation to find the angle β

$\beta = \sin^{-1} y1/z$ …… (2)

or $\beta = \cos^{-1} x1/z$ …….. (3)

β is the angle that the robot must turn to stand on the beginning of the Z line.

3.3.1.3 Calculate the shortest distance to the goal

To calculate the distance between the robot and goal with availability of the obstacles follow these steps:

1- After determine how long is the Z line and how is the β angle, the robot turns by the (β angle), then ultrasound sensor checks the availability of the obstacles on the road, if there's no obstacles the robot moves straight to the goal directly, else if the robot detects the first obstacle after specific distance the robot goes forward distance which is implemented in L1. As in *Figure 3.15*, then its stops before the first obstacle. In our algorithm the robot stops before the first obstacle with supposed distance (equal 10 cm) at the first point (p1), then the ultrasound sensor turn 45° to the left and 45° to the right to detect the suitable path that the robot will take to turn around the obstacle see *Figure 3.16.*

2- After the robot decisions the suitable path, it will turn about 90° from the p1, then it moves 20 cm forward, after 20 cm it will stop then turn again 90°, then its moves forward about 20 cm around the obstacle this distance called D1, then the robot turn 90° again and move forward 20 cm also, after that it turns 90°, here the robot will return to the Z line with the same angle of the start position (β angle) at the second point P2. As in *Figure 3.15*.

3- The robot will repeat the detection of obstacles avoidance again from P2. In our example there are two obstacles so the robot repeats the previous operation.

Where the robot moves L2 distance and stops at P3, then turn around the second obstacle until it reaches the point four P4 here the robot detections the availability of the obstacles. When it finds no obstacles so it will move directly the distance L3 to reach the goal, in *Figure 3.15* the robot calculate L3 by using these equations :

Z = L1+D1+L2+D2+L3 ……..(4)

L3 = Z – (L1+L2+D1+D2) ……..... (5)

D1 = D2= supposed distance 20 cm

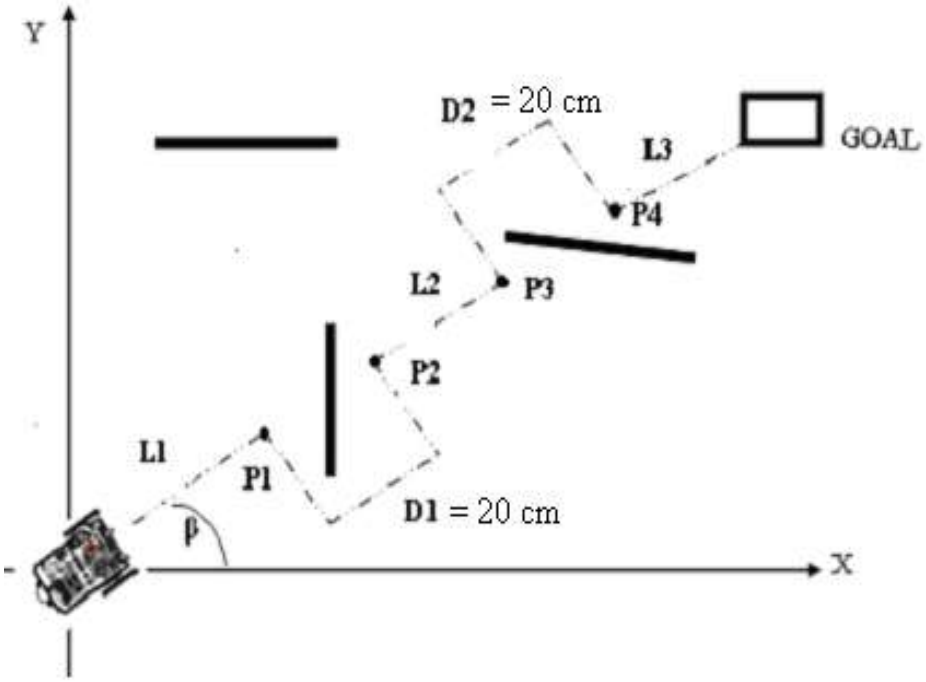


Figure 3.15 The path that the robot is expected to follow to the goal using the shortest distance approach given the indicated three obstacles

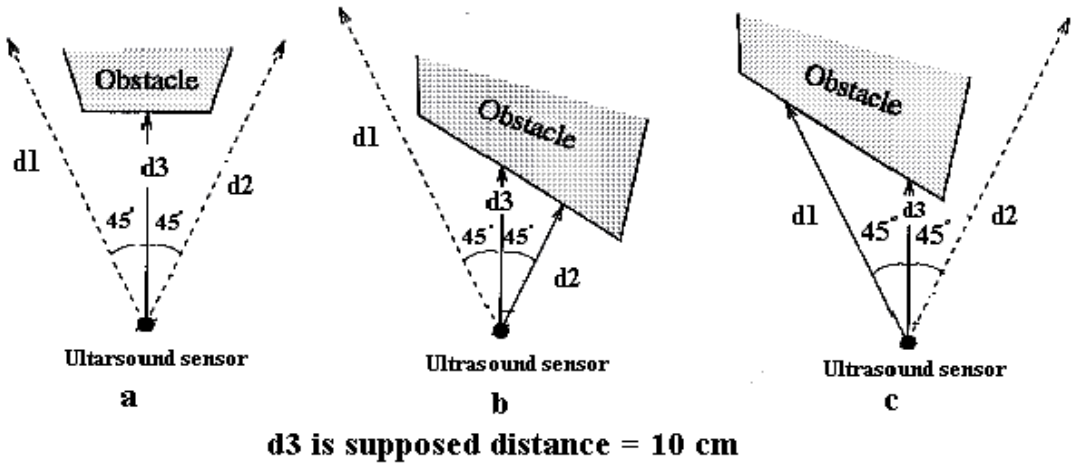


Figure 3.16 Avoiding the obstacles using ultrasound sensor

In *Figure 3.16* (case a) the ultrasound sensor turn 45◌̊ to the left and 45◌̊ to the right to detect the obstacle and choose the suitable and safe path , in this case it detects no obstacle in the left and in the right side, so the robot takes any path d1 or d2 to turn around the obstacle. In algorithm of obstacle avoidance it takes the right side so its moves the d2 distance.

In (case b) the ultrasound sensor turn 45◌̊ to the left and 45◌̊ to the right, then it detects the obstacle in the right side, so the robot takes the left side and moves the distance d1 to turn around the obstacle.

In (case c) the ultrasound sensor turn 45◌̊ to the left and 45◌̊ to the right, then it detects the obstacle in the left side, so the robot takes the right side and moves the distance d2 to turn around the obstacle.

3.3.1.4. The Algorithm

As seen in previous *Figures (3.14, 3.15, 3.16)* the algorithm that used for Boe-Bot navigation for one or N obstacles to reach the goal's location is:

1- Firstly determine the goal's location (x, y) coordinate where the Boe-Bot location (0, 0) coordinates.

2- After that find the shortest distance between the robot and the goal (Z) using $Z^2 = x^2 + y^2$

3- Next find the angle (β). That the robot must turn to be on the beginning of the shortest line (Z) using trigonometric equation $\cos^{-1}(x/z) = \beta$.

4- Then by using ultrasound sensor the Boe-Bot can detect the obstacles (in case obstacles is available) that the robot may avoid it during travel to the goal and determine the exact distance L1 between itself at the start position (0, 0) and the obstacle. In the case of the obstacles are unavailable the robot will go directly to the goal.

5- Then the robot starts to travel L1 distant, after it finishes L1, stops before the obstacle with supposed distance (10 cm) at specific point.

6- After the robot stops the ultrasound sensor checks (45°) left and (45°) right to find the suitable and safe path to turn around the obstacle see *Figure 3.16*.

7- After the robot takes the suitable path to turn around the obstacle. It returns to Z line at the point (P1) with the same β angle as in *Figure 3.15*.

8- Then the algorithm will be repeated again starting from the fourth step of the algorithm, until the robot reaches point P4, here the robot determines the distance to the goal L3, where $L3 = (Z - (L1+D1+L2+D2))$. Assume that D1=D2 = 20 cm.

The implementation of the above algorithm is given in Appendix A (Program 8).

3.3.2 Algorithm of path tracking Boe-Bot robot

This work describes a path tracking method for wheeled mobile robots. To move a mobile robot along a desired path, two variables must be controlled, namely position and orientation. The proposed path tracking method has been implemented. The performance of this algorithm by simulation and experimental results are shown. For navigation photoresistor sensors are used with pointing ahead on the ground in front of the Boe-Bot.

3.3.2.1 Methodology

The method used in this work depends mainly on the photoresistor sensor and its capability of measuring the quantity of reflected light from onboard and measuring RC decay time. This work was done using two different methods.

The first approach involves the light coming from different sources (lamps, sunlight) in the environment as indicate in *Figure 3.17*, but this method has some drawbacks. The first drawback is that the environment light is changeably relative to the location and the strength of the light source.

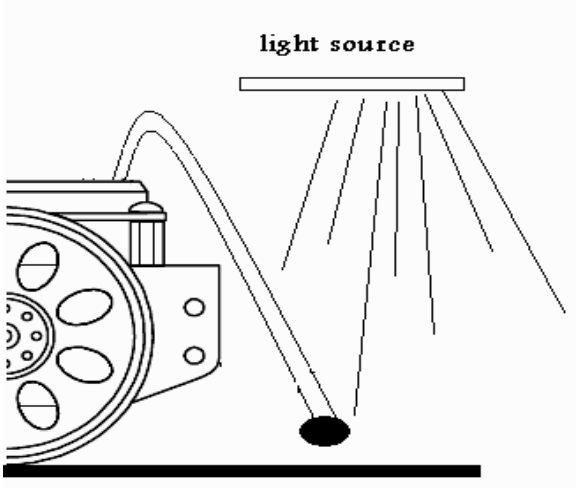


Figure3.17 Effect of the external light sources on the photoresistor

The second drawback is the shadow of the robot itself, where the sensor data differs if the sensor lies under robot's shadow from it does not. So the sensor will generate faulty data as demonstrated in *Figure 3.18*.

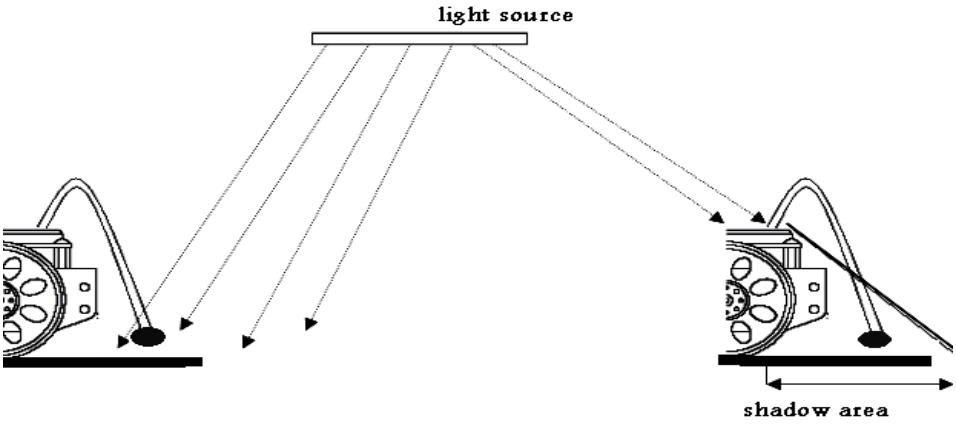


Figure 3.18 The robot's shadow

The second approach is to give the robot a separated, isolated and unchangeable light source, which moves with the robot to avoid the drawbacks in the first approach given in *Figure 3.19.*

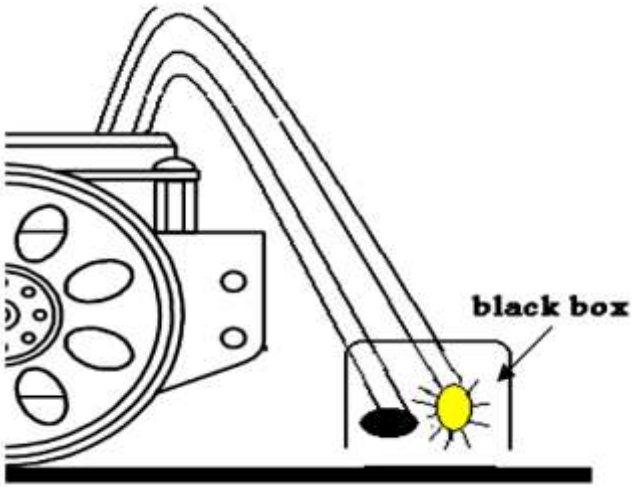


Figure 3.19 Isolated light source with photoresister sensor

3.3.2.2. The Algorithm

Suppose we have black track on white board as in *Figure 3.19* and the robot will travel over it. The maximum width of the track is limited by the width of the used robot, the maximum width of the track that can be used in this algorithm is 7 cm, and the minimum width is 2 cm, minimum width was found experimentally. The algorithm is:

1- Photoresistor sensors check the location of the robot over the board.
2- If the right and the left photoresistor sensors are over the white area the robot will go forward.
3- Else if the right photoresistor sensor detects white area and the left sensor checks black track, the robot will turn to left by 4.5 degrees (note: this angle was the most suitable angle experimentally found)
4- Else if the right photoresistor sensor detects blacks track and the left sensor checks white area, the robot will turn to right by 4.5 degrees.
5- Else if both of sensors over black track, robot will stop (end of track).

The implementation of this algorithm is given in Appendix A (Program 9).

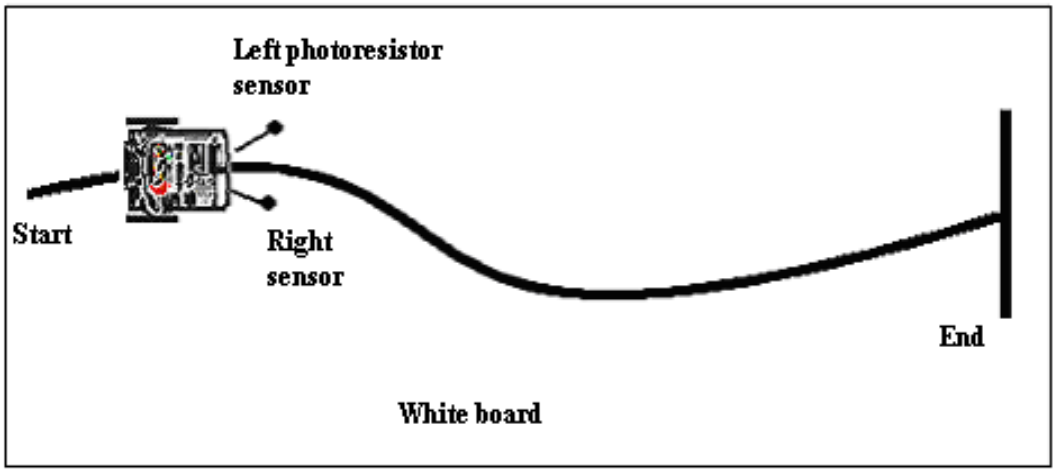


Figure 3.20 Path tracking example

3.4. Summary

Frequency sweep was introduced as a way of determining distance using the Boe-Bot's IR LED and detector. FREQOUT was used to send IR signals at frequencies ranging from 37.5 kHz (most sensitive) to 41.5 kHz (least sensitive). The distance was determined by tracking which frequencies caused the IR detector to report that an object was detected and which did not. Since not all of the frequencies were separated by the same value, the LOOKUP command was introduced as simple way to use the counting sequence supplied by a FOR…NEXT loop to index sequential lists of numbers.

Control systems were introduced along with closed-loop control. In proportional control in a closed-loop system, the error is multiplied by proportionality constant to determine the system's output. The error is the measured system output subtracted from the set point. For the Boe-Bot, both system output and set point were in terms of distance. The BASIC stamp was programmed in PBASIC to operate control loops for the both the left and right servos and distance detectors. By re-sampling distance and adjusting the servo output before sending pulses to the servos, the control loop made the Boe-Bot responsive to object motion. The Boe-Bot was able to use proportional control to lock onto and follow objects, and it also used it to track and follow a stripe of black electrical tape.

In this chapter, algorithms for obstacle avoidance and path tracking were discussed. Determining angle and distance variables were needed to get accuracy measurement to achieve safety navigation without obstacles collision. And path tracking algorithm keeps the correct track of robot without outside light effective.

CHAPTER FOUR. DEVELOPMENT OF THE OBSTACLE AVOIDANCE AND PATH TRACKING SYSTEM FOR THE MOBILE ROBOTS

4.1. Overview

In this chapter the implementation of navigation and path tracking system is considered. The relations between system components (sensor, microcontroller, servo motor, wheels, and interface) were discussed. Flow charts of obstacle avoidance and path tracking mobile robot algorithm were presented.

A navigation technique with obstacle avoidance is proposed for mobile robots in which the dynamics of the robot are taken into consideration. The information needed is the (x,y) coordinate of the goal. The navigation to the goal and obstacle avoidance is achieved by switching the direction angle of the robot. The effectiveness of the technique is demonstrated by simulation.

Simulation is often used as part of the development work for autonomous vehicles and robots in general. However, it is commonly believed that the physical interaction between robot and environment is not possible to simulate in a sufficiently accurate manner. The modeling of the physical interaction between wheels and ground and also the robots dynamics is done. However, it is difficult to transfer simulation results to the real-life physical domain. The major problems simulations of out-door robots encounter are the following:

• Numerical simulations do not usually consider all physical laws of the interaction between robot and environment, such as mass, weight, friction, inertia, etc.

• Physical sensors are noisy and actuators have uncertain effects. This is seldom modeled in simulation since it requires exact knowledge of the nature of all these uncertainties.

• Physical sensors and actuators differ because of wear and tear, differences in manufacturing, and differences in their position on the robot and because of varying weather or environmental conditions.

4.2. Structure of the navigation system

Navigation system is designed to make the robot meets the requested tasks. In general, a navigation system can be represented by the block diagram shown in *Figure 4.1*.

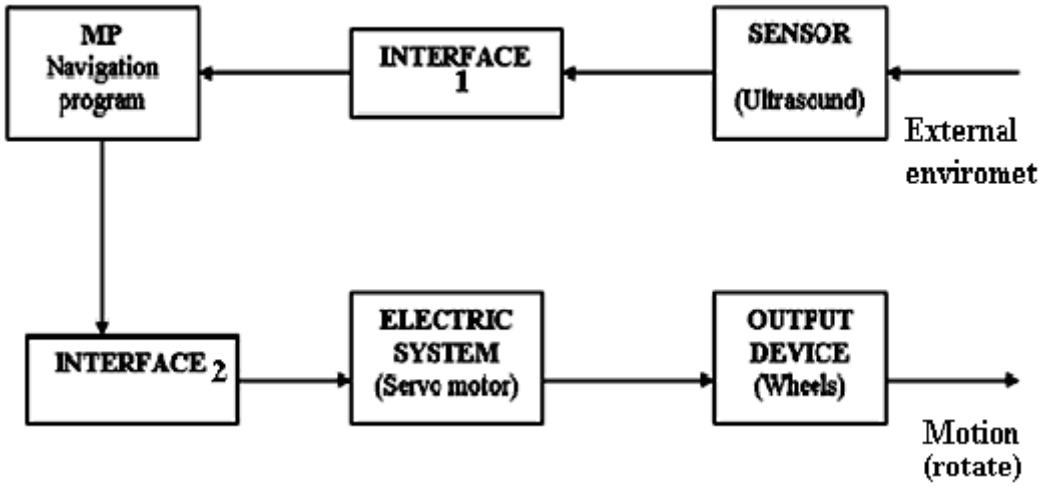


Figure 4.1 Block diagram of the navigation system

Sensors make the robot part of the environment in which it exists. They give the robot information about the work cell that is vital to normal operation. Sensors are used to (1) protect worker and robot from harm, (2) monitor the production system and work cell operation, (3) analyze product quality, (4) provide part identification and orientation. All sensors are grouped into either a contact or non contact category, and each type can have either discrete or an analog output signal. Contact sensors include limit switches and artificial skin, and the most commonly used non contact sensors include proximity sensors, photoelectric device, and range finders are used to find larger distances, to detect obstacles, and to map surface of objects. Range finders are meant to provide advance information to the system. In the navigation system type of range finder sensor which is ultrasound sensor is used based on sending ultrasound signal to output environment to detect available of objects then receiving the echo which reflects from

the objects. Most ultrasonic sensors measure the distance using the time-of-flight technique. In this technique, a transducer emits a pulse of high frequency ultrasound which travel a certain distance and is reflect back when it encounter a separation in the medium, it is then received by receiver . The distance between the transducer and the object is half the distance traveled, which is equal to the time-of-flight times the speed of sound. Sensor can provide simple, analog information about the position of obstacles. This information must be digitized before microcontroller can use it [11, 12].

Using interface the component of the navigation system will be able to interact together. Interface is defined as a place at which independent system meet and act on or communicate with each other. In our system the sensor supplies data to the brain of the system microprocessor through interface 1 which is analog-to-digital converter (ADC) built into MC**.** Microcontroller reads and processing the digital data. After MC processing data it sends the result to the servo motor by using cable which consider as interface 2 connected to the pin 12 and pin 13 of MC. Electric system includes a source of electric power and an electric motor, in our robot the motor is a servo motor a precision electric motor that causes rotational motion in proportion to supply electrical command signal from a servo amplifier. DC servos are used primary for higher power application where larger torque is necessary. AC servo motors tend to be more stable in their operation, lighter in weight, more rugged, and required less maintenance. An important issue in all electric motors is the back electromotive force, or back emf. A Wire carrying a current within a magnetic field will experience a force, which causes it to move. Similarly if a wire (conductor) moves within a magnetic field such that it will cross the field line, a current will be induced into the conductor this is the basic principle of electric power generation. However, it also means that when the wires of the windings in a motor a rotating in the magnetic fields of the magnets, a current will be induced in them in the opposite direction of the input current. This current is called back-emf, and it tends to reduce the effective current of the motor. The faster the motor rotate. Rotating the servo will cause to rotate peripheral device called wheels which are the direct reason to make the robot interact with outside environment [11].

4.2.1. Flow chart of obstacle avoidance algorithm

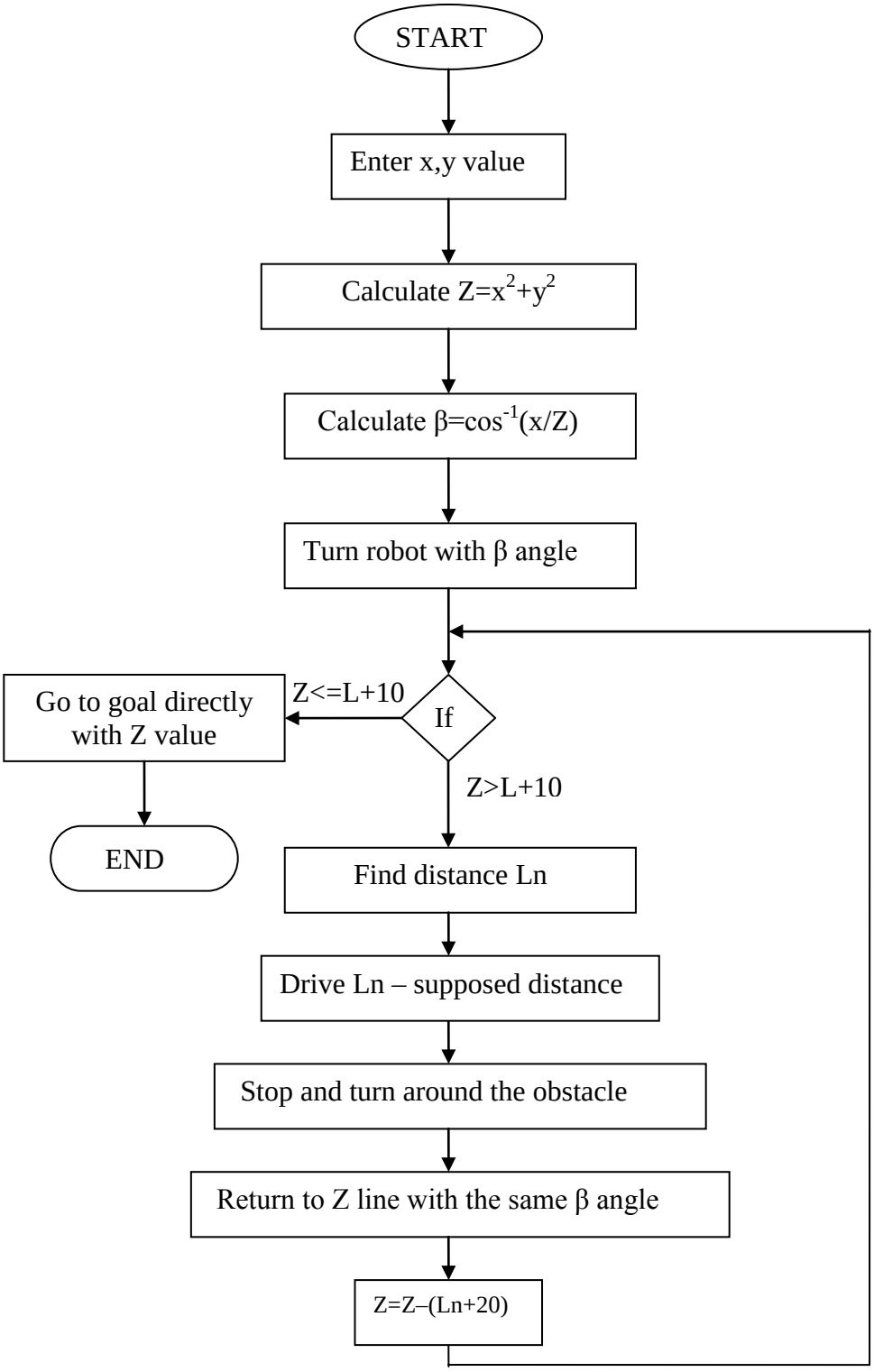


Figure 4.2 Flow chart of the obstacle avoidance algorithm

The previous flow chart can be described through these steps:

1- Input the coordinates (x, y) for the goal.

2- Using (x, y) coordinates to find Z by ($Z^2 = x^2 + y^2$), then find β angle by

($\cos^{-1}(x / Z) = \beta$).

3- Using ultrasound sensor to find the distance between the robot and the first obstacle.

4- The robot moves distance Ln.

5- The robot will rotate around the obstacle then return to the goal line.

6- The operation will be repeated in the case of the obstacle availability

7- Else the robot will move directly to the goal

4.2.2. Flow chart of the path tracking algorithm

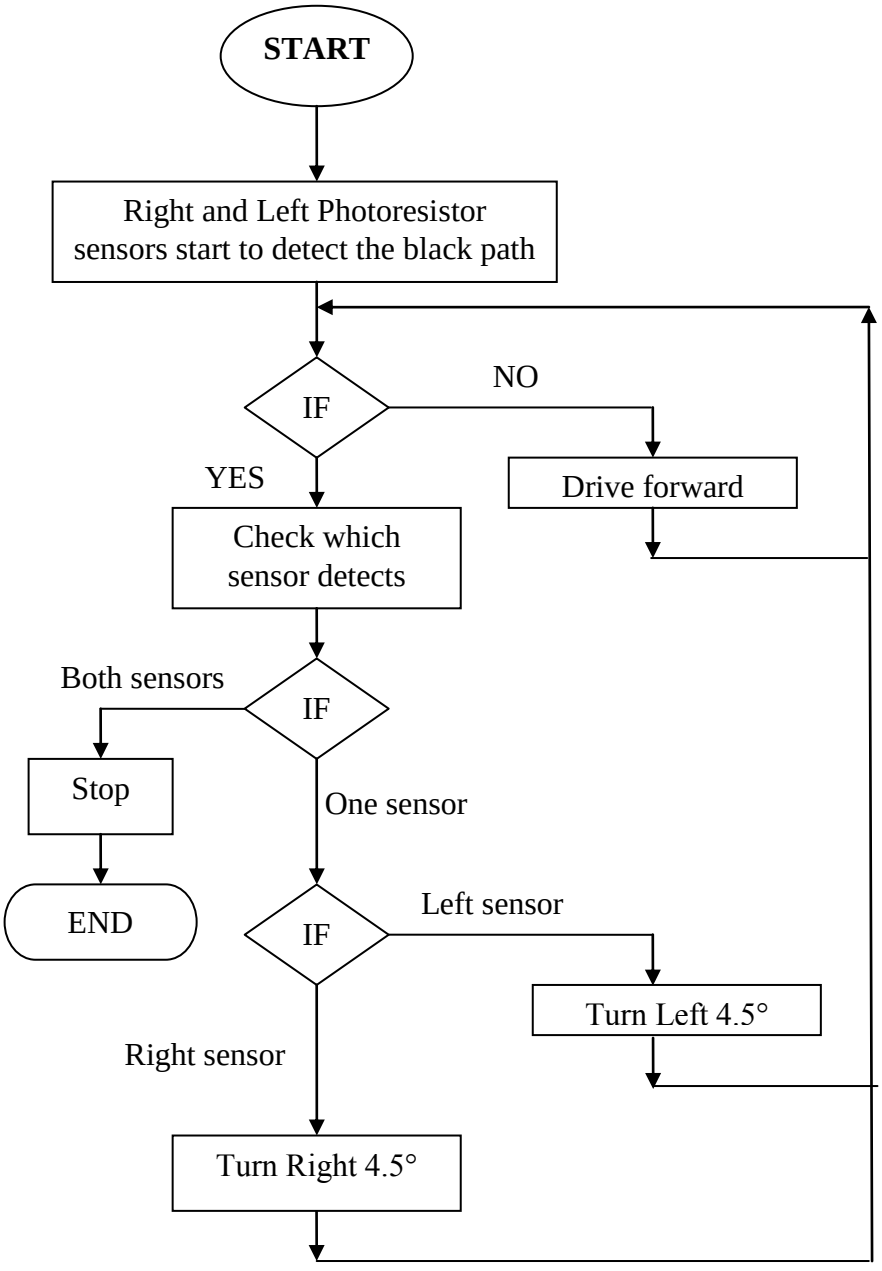


Figure 4.3 Flow chart of the path tracking algorithm

The previous flow chart can be described through these steps:

1- The photoresistor sensor starts to detect the black track.

2- If the left and right photoresistor sensors on white area, the robot will go forward.

3- If the left photoresistor sensor on the white area and the right sensor on the black track, the robot will turn to the right.

4- If the left sensor on the black track and the right sensor on the white area the robot will turn to the left.

5- If the left and right photoresistor sensor on black track, the robot will stop.

4.3. Simulation and practical results of the Boe-Bot robot navigation

In this section two simulation example are given for the Boe-Bot navigation algorithm that was developed in this thesis, after practical results were practically found. The first one will be about obstacles avoidance algorithm, the second about path tracking algorithm.

Practical results were found by fixing pencil on the Boe-Bot robot to mark the way that the robot actually navigates, as shown in *Figure 4.4*, by follow the marked line during robot's navigation, practical path of robot can be found.

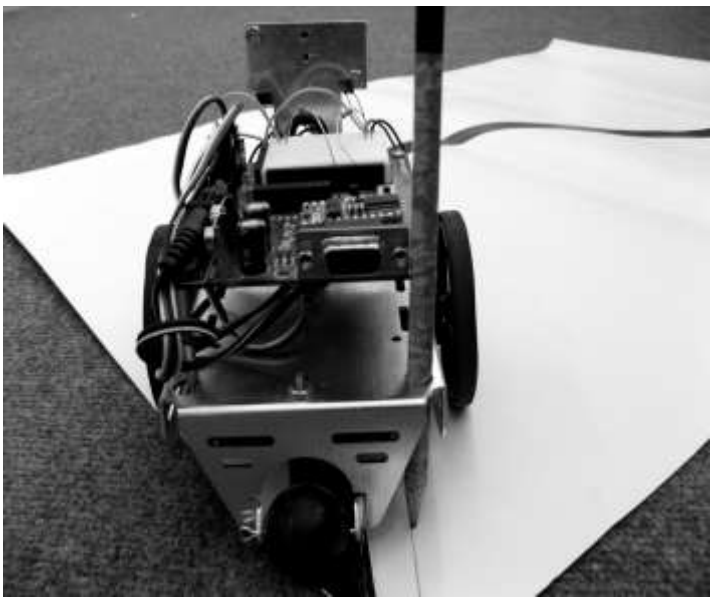


Figure 4.4 Obtaining practical results by using a fixed pencil with the robot through its motion

4.3.1. Obstacle avoidance simulation

In this simulation example three cases were discussed depending on the number of the obstacles in the robot's path.

Case 1: Navigation without obstacles

Figure 4.5 below shows the coordinate of the goal and robot. Where the robot is placed on the origin (0, 0) in the direction of the x axis. And the goal is placed at the (75, 30), using this coordinate, the robot can calculate Z distance. Then β angle.

Distance $Z = (75^2 + 30^2)^{½} = 80.8$ cm. This is the shortest distance between the robot and the goal.

Now find the angle that the robot will turn to face the goal's direction (β) by using

$\cos^{-1}(75/80.8) = 21.8$ degrees

After that the robot rotates 21.8 degrees from the origin position, then the ultrasound sensor start to detect the availability of the obstacles. In this case there are no obstacles so the robot will go directly about 80 cm to the goal, the Z = L1 = 80 cm. The result can be summarized in *Table 4.1*.

Table 4.1 Summary of the simulation result of case1

X	Y	Z	β	Distance to the goal = L1 (cm)
75	30	80.8	21.8°	80.8 cm

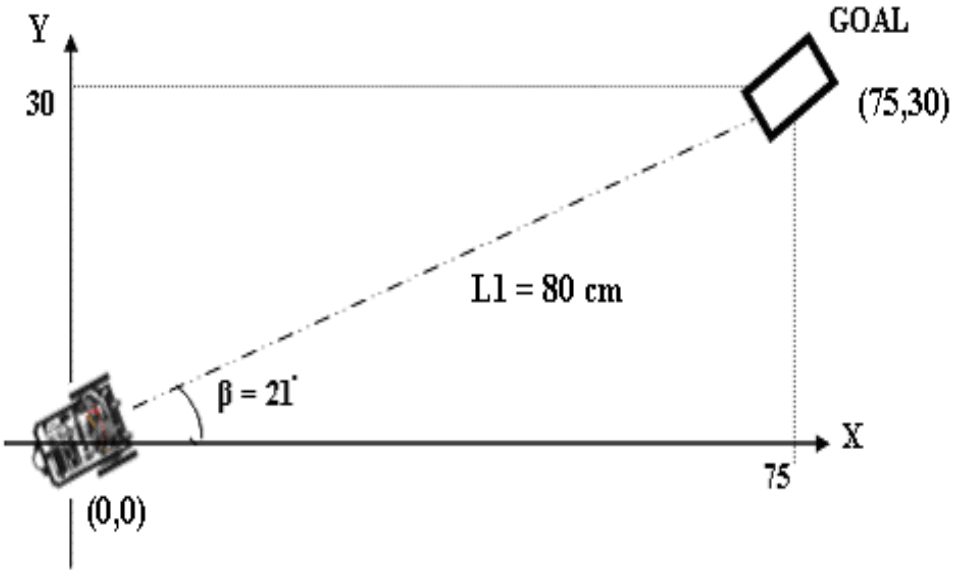


Figure 4.5 Navigation without obstacles

Practical results of Case 1:

Practically as shown in *Figure 4.6* where 1 is the expected path and 2 is the actual path, the values of parameters were:

- β = 22 degrees
- Z = 75 cm
- The end goal's coordinates (72, 23)

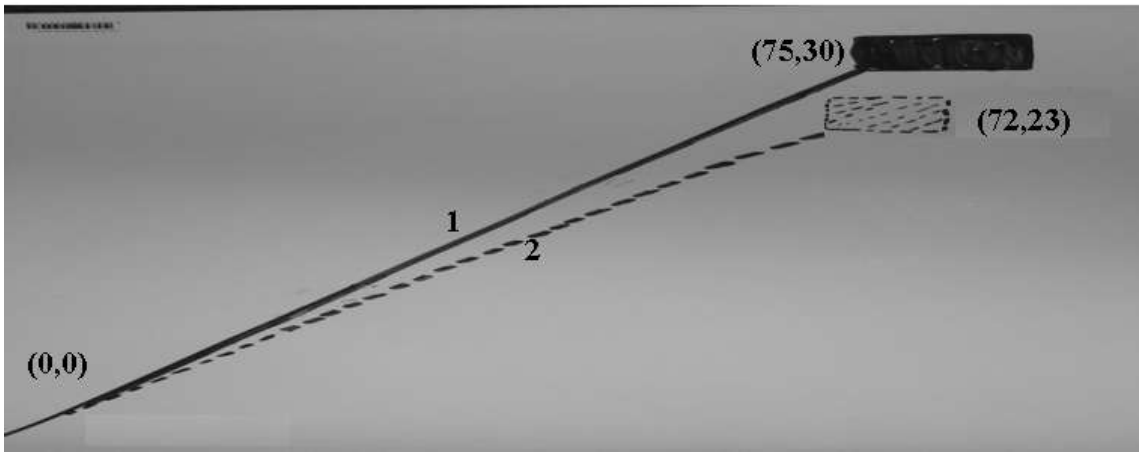


Figure 4.6 Actual image of the results obtained for Case 1

Case 2: Detection one obstacle

Figure 4.7 below shows case 2, robot in (0, 0) in the direction of the x axis, and the goal's coordinate is (80, 80).

The distance $Z = (80^2+80^2)^{½} = 113$ cm

And the β angle = $\cos^{-1}(80/113) = 45$ degrees

In case 2 the ultrasound will find obstacle after 65 cm, so the robot will move L1 distance which is 55cm. Then stops at point 1 (P1) before the obstacle with assumed distance = (10 cm), Then the ultrasound sensor checks 45 degrees left and 45 degrees right to take the suitable path to turn around the obstacle, After the robot turns around the obstacle stops at point 2 (P2) with (β) angle, after that the robot repeats detection operation. In this case there's no other obstacles so the robot goes directly to the goal after its calculate L2, where $L2 = Z - (L1+20)$, (20 comes from 10 cm × 2).

L2 =113 – (55+20) = 113 – 75 = 38 cm distance from P2 to the goal.

Table 4.2 summary of the simulation result of case 2

X	**Y**	**Z**	**B**	**L1**	**L2 = Z – (L1+(20×1))**
80	80	113	45°	55	113 – (55 + 20) = 38 cm

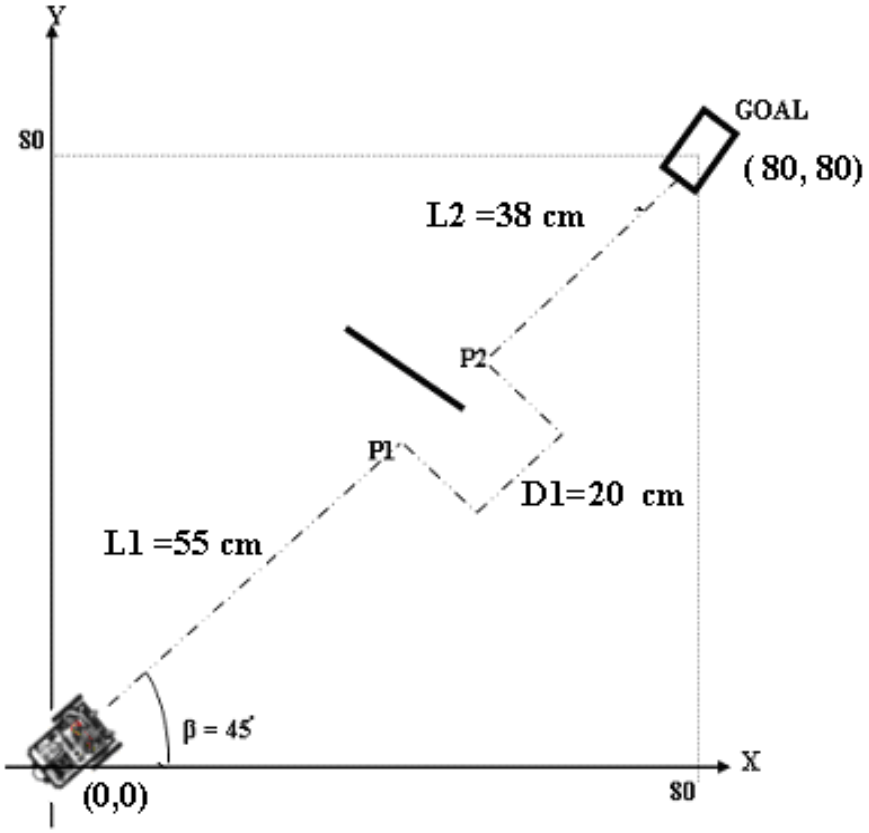


Figure 4.7 Navigating with a single obstacle

Practical results of the case 2:

Practically as shown in *Figure 4.8* where 1 is the expected path and 2 is the actual path, the values of parameters were:

- β = 45 degrees
- Z = 104 cm
- L1 = 52 cm , D1 = 19 cm
- L2 = 33 cm
- The end goal's coordinates (74, 70)

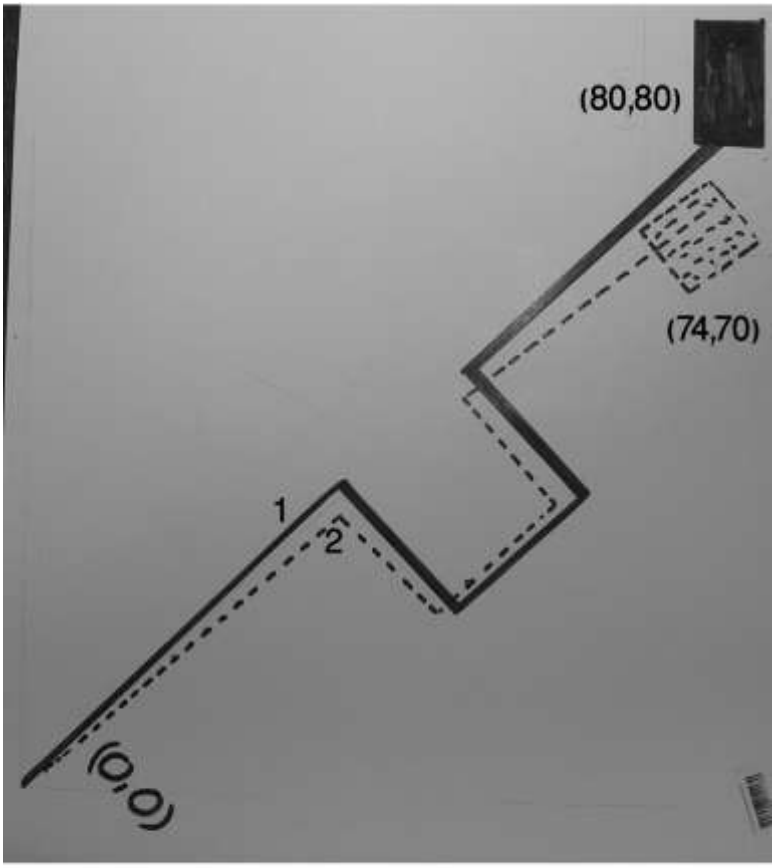


Figure 4.8 Actual image of the results obtained for Case 2

Case 3: Navigation with N obstacle

In this example, there are (N) obstacles and the goal has (50,150) coordinate as shown in *Figure 4.9* below.

The distance $Z = (150^2+50^2)^{½} = 158$ cm

The β angle = $\cos^{-1}$ 50/158 = 71 degrees

In case 3 the robot follows the same procedure in case 2 until it reach the last obstacle (N). Here the robot calculates the distance L_{N+1} between the robot and the goal by:

$$L_{N+1} = (Z - (L1+L2+L3+\ldots\ldots\ldots.+ 20 \times N))$$

For example: three obstacles (N=3) were found during the path. The robot at (0, 0) and in the direction of x axis, the robot detect the availability of the first obstacle after 20 cm from the (0, 0) location, so L1=20-10=10 cm. Now between the robot at P2 and the second obstacle = 36 cm so L2 = 36 – 10 cm = 26 cm. And between the robot at P4 and the third obstacle which is the last one = 15cm so, L3 =5cm. Where D1, D2, D3 = 20

cm. At the P6 the robot detection there's no obstacle in front it so it will go directly to the goal after it measures the L4 distance, L 4= (158 – (10 + 26 + 5 + (20) (3)) = 158 – 101 = 57 cm between the robot at P6 and the goal.

Table 4.3 summary of the simulation result of case 3

X	Y	Z	β	L1	L2	L3	L4 =(Z – (L1+L2+L3+(20)(3))
50	150	158	71°	10	26	5	(158 – (10+26+5+60)) = 57cm

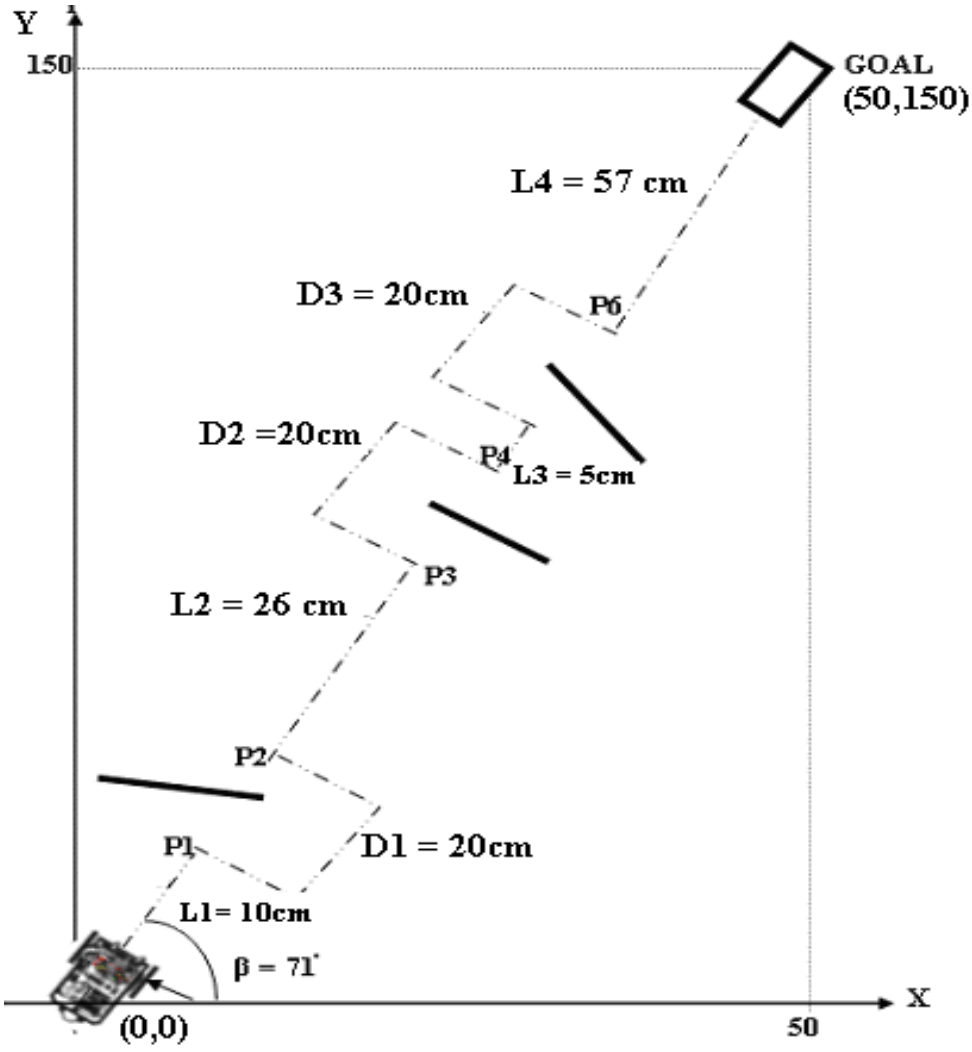


Figure 4.9 Navigation with multiple obstacles

Practical results of the case 3:

Practically as shown in *Figure 4.10* where 1 is the expected path and 2 is the actual path, the values of parameters were:

- β = 68 degrees
- Z = 147 cm
- L1 = 10 cm , D1 = 19 cm
- L2 = 24 cm , D2 = 19 cm
- L3 = 5 cm , D3 = 19 cm
- L4 = 51 cm
- The end goal's coordinates (44,141)

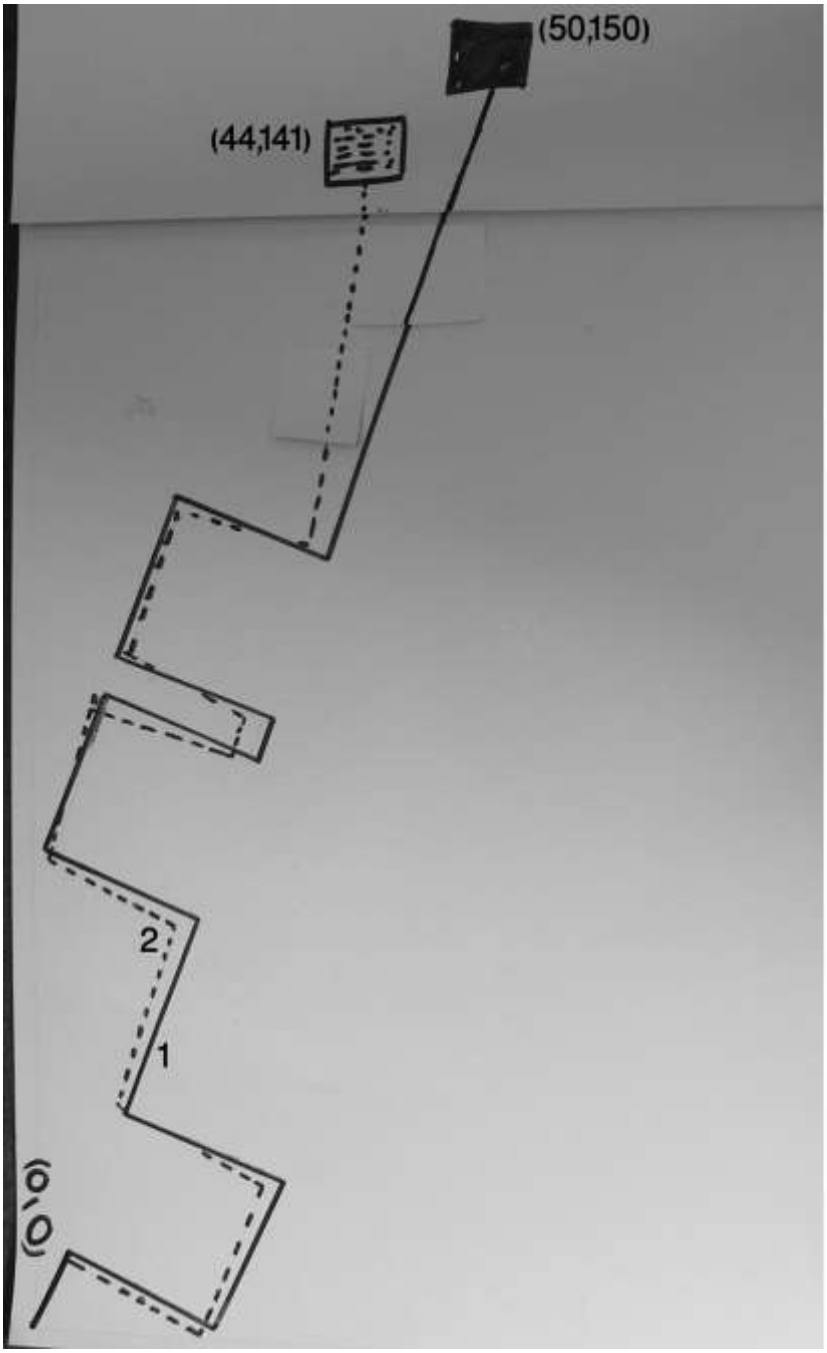


Figure 4.10 Actual image of the results obtained for Case 3

4.3.2. Path tracking simulation

Figures below show how the both side of the photoresistor sensor detect the white board or the black track, and the robot will navigate on this track depending on the detection of the both side of the photoresistor sensor, (the width of this track = 7 cm).

1- *Figure 4.11* shows the start of the track, the both sensors detect the white board, so the robot will go forward.

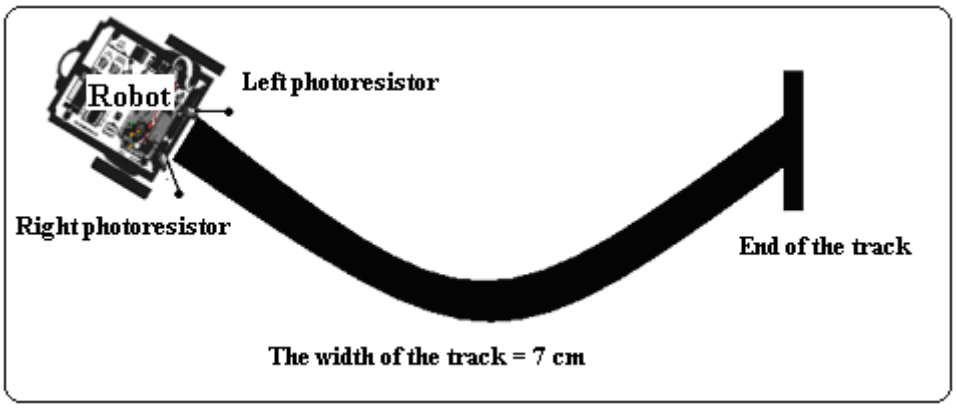


Figure 4.11 Both photoresistor sensors detect the white area. Therefore the Boe-Bot will continue to move forward.

2- *Figure 4.12* shows that the left sensor detects the black track, so the robot will rotate 4.5 degrees to the left.

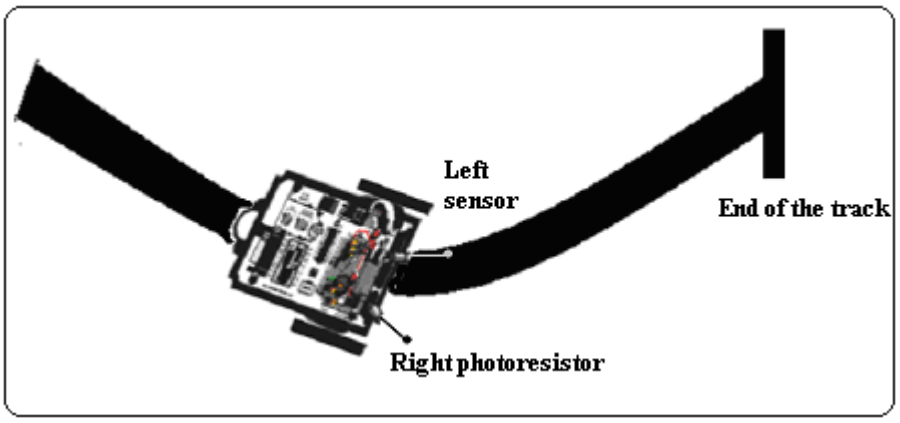


Figure 4.12 Left photoresistor sensor detects the black track. Therefore the Boe-Bot turns left.

4- *Figure 4.13* shows that after the previous state the robot traveled forward until the right sensor detect the black track, so the robot will rotate to right 4.5 degrees.

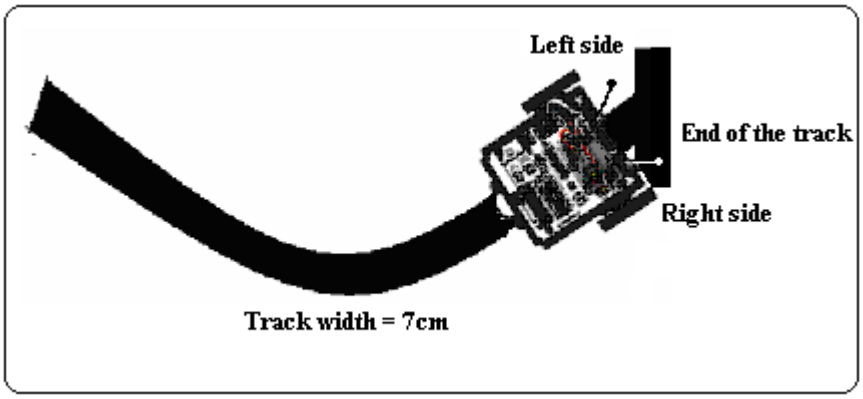


Figure 4.13 Right photoresistor sensor detects the black track. Therefore the Boe-Bot turns right.

4- The last *Figure 4.14* shows that both sensors detect the black track, which means end of the track, so the robot will stop.

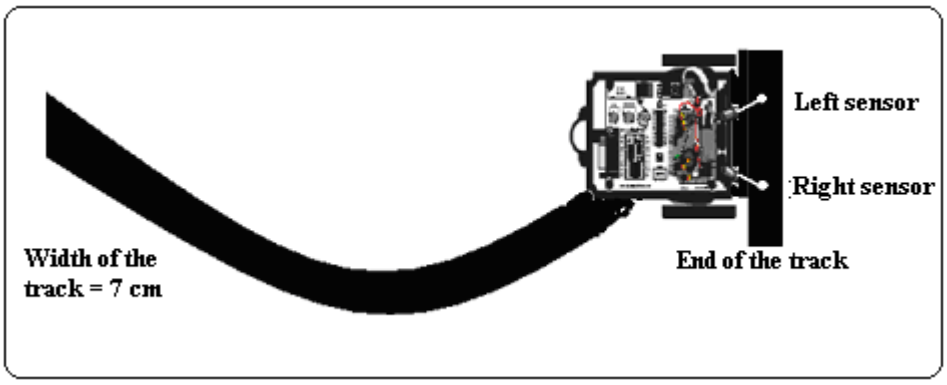


Figure 4.14 Both photoresistor sensors detect the black track that indicates the end of the track. Therefore the Boe-Bot stops.

Practical results of the path track algorithm:

Figure 4.15 shows the practical path of the robot that uses the algorithm of the path tracking. Where the thin line is the practical line.

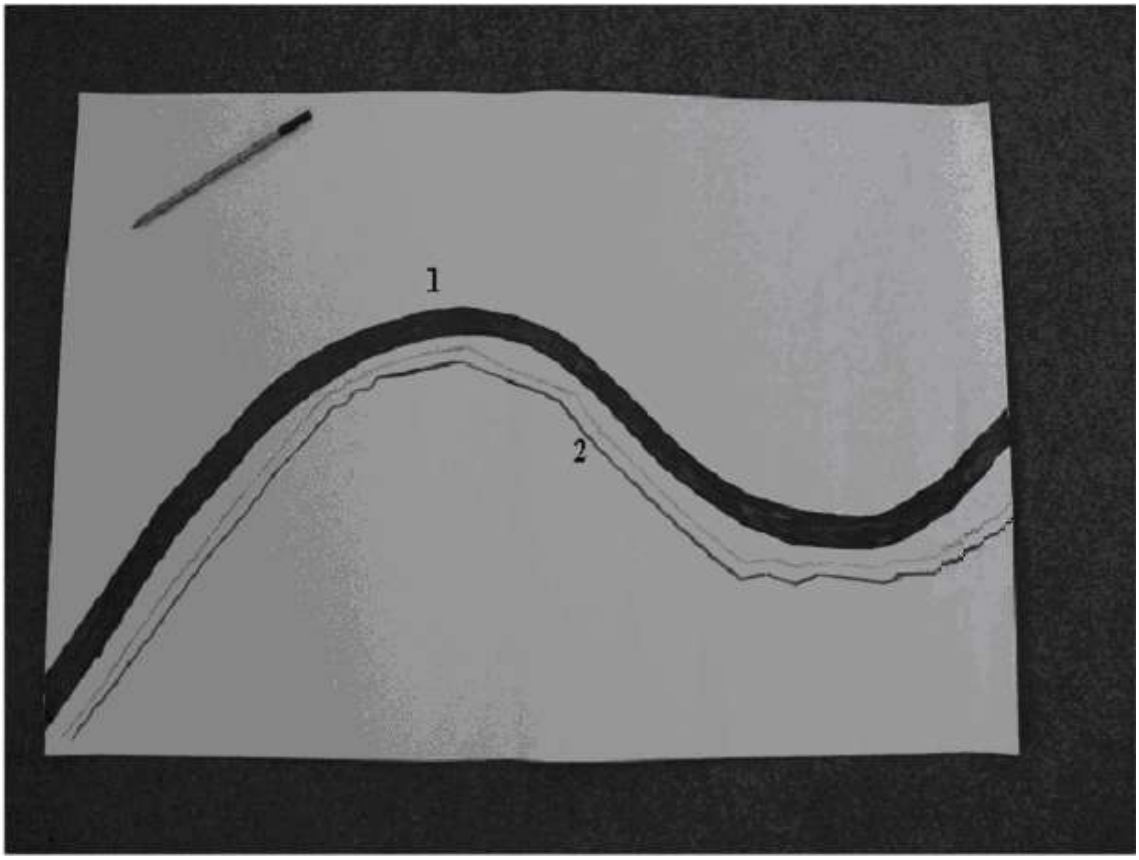


Figure 4.15 The practical result of the path tracking algorithm: black line 1 is path, line 2 is robot track drawn by marker

In *Figure 4.15*, number 1 is the desired path which drawn by user, and number 2 is the track of the Boe-Bot robot.

After comparing between simulation and practical work for both obstacle avoidance and path tracking algorithm, reasons of differences results are:

- The main problem in Boe-Bot Robot is that the processor used (BS2) has not the ability to deal with the floating point numbers, this problem caused some limitation for this project as will be discussed.

- The Boe-Bot robot has two servo motors (one that steers by changing the speed/direction of the two side wheels) it tends not to drive in a straight line. This is due to the two motors not quite reaching the same speeds; hence it will describe a curve, as a final result not to drive in straight line will decrease the actual distance and change the desired position.
- The practical result of the path tracking algorithm is changeable with light conditions around the robot. So to get a good result in this algorithm, the experiment must be done in area has enough unchangeable light sources.

4.4. Summary

The structure of navigation systems is given. The components of the system are unified inside the system. Sensor is component that connect and transfer the data from the external environment to the robot in different way depend on the type of sensor. They used to measure the distance. This information is supplied analog data to MC which digitize this data before send it to servo motor through cable. Lastly the processing data converted to movement through the wheels.

In path tracking algorithm sensitivity of photoresistor sensor, light sources from the external environment, RC decay were information must be consider in path tracking algorithm. Simulations examples results for the obstacle avoidance and path tracking algorithm were given. Then compare them with the practical results. And conclusion the reasons of these differences.

5. CONCLUSION

In this theses path tracking and obstacle avoidance problem of mobile robot are considered. Autonomous mobile robot reaches destination point without human intervention in unknown environment. The control structure of mobile robot for path tracking and obstacle avoidance are presented. Electronic schemes are designed and computer programs are developed using Basic Stamp Editor. In pathtracking problem using photoresistor sensor mobile robot follows the road drawn by the user. Photoresistor sends input sensor signal to the microprocessor, the developed program on the base of the input signal controls the servos.

The navigation and obstacle avoidance of mobile robot was solved by using tactile, infrared and ultrasonic sensors. Hardware schemes and software are developed for navigation. The developed programs are loaded to microprocessor of mobile robot. Developed software on the base of sensor signal analyzes the environment. In case of availability of obstacle, it sends corresponding control signals to servo motors of the wheels to avoid from obstacle.

The hardware and software that were developed within this thesis allow the controlling and navigating mobile robots in an uncertain environment. Simulation and practical work for the obstacle avoidance and path tracking algorithm were done. The implementations of developed algorithms and software have been done in Parallax Boe-Bot robot.

ABSTRACT

Path tracking and obstacle avoidance are two very important issues that must be considered in the process of developing robot navigation systems. In this thesis, a description of obstacle avoidance and path tracking problems of mobile robots is given. Hardware and software components of the Boe-Bot robot are described. The hardware scheme and the software implementation involves a collection of activities that are discussed in this thesis. The thesis begins with an introduction to the Boe-Bot brain, BASIC Stamp microcontroller, programs for the Boe-Bot basic maneuvers, and then it proceeds with the construction of electrical circuits by using different types of sensors (whiskers, photoresisror, infrared, and ultrasound). The hardware circuits were designed on the board of the Boe-Bot robot for obstacle avoidance and path tracking using mechanical, infrared, and ultrasound sensors. Each design was implemented and tested in the environment with physical obstacles to perform autonomous tasks. The implementation of the obstacle avoidance and path tracking algorithms for mobile robots are discussed. The simulation and practical work of the obstacle avoidance and path tracking algorithm are done for a variety of cases. The programs that implement the algorithms for path tracking and obstacle avoidance were written in PBASIC using the Basic Stamp Editor, and the programs were executed on the BS2 (Basic Stamp) microcontroller on the Parallax Boe-Bot mobile robot.

ACKNOWLEDGMENT

ALLAH who definitely is not required any thanks or gratitude; my extreme thanks and gratitude's are deeply introduced from my hearts, hoping, to who will help me to be good by following and his guidance.

I would like to acknowledge the many people without whom I could not complete this thesis. First of all, I thank my advisor, and Asst. Professor Rahib Abiyev for his guidance throughout the thesis. His help throughout the writing was invaluable. And I would like to thank all committee members of Computer Engineering Department at Near East University and all of people I have there support through all my studies. Finally, with all of my love I would like to thank my husband for his support through all my studies and my family for there encouraged and patient throughout all years.

TABLE OF CONTENTS

LISTS OF FIGURES

LIST OF TABLES

6. REFERENCE

[1] James A.Rehy. "Introduction to robotics in CIM system" 2000, 2003 Pearson education, Inc. Pearson Prentice Hall upper saddle river, New jersey 07458

[2] Roland Siegwart & illahR.nourbakhsh "Autonomous mobile robots." Proc. of the IEEE International Conf. on Robotics and Automation, Barcelona, Spain, pp1817-, 2005.

[3] Melo, F.A., Lima, P., Ribeiro, M.I.: "Event-driven modeling and control of a mobile robot population". Proceedings of the 8th Conference on Intelligent Autonomous

Systems, Amsterdam Netherlands (2004)

[4] Simmons, R., Koenig. "Probabilistic Robot Navigation in Partially Observable Environments". Proceedings of the International Joint Conference on Articial Intelligence, Montreal, Canada (1995) 1080_1087.

[5] O. Khatib. "Real-time obstacle avoidance for manipulators and mobile robots" International Journal of Robotics Research, Vol. 5, No. 1, pp. 90-98, 1986

[6] Munsang Kim, and Chongwon Lee Woojin Chung. "Integrated navigation system for indoor service robots in large-scale environments," Proc. of the IEEE International Conf. on Robotics and Automation., USA, pp.5099–5104, 2004.

[7] K. Konolige. "A Gradient Method for Real time Robot Control" Proc. of the IEEE/RSJ Conf. on Intelligent", Japan, pp. 3450–3455, 2004.

[8] Courtney, J.,"Mobile Robot Localization Using Pattern Classification Techniques," M.S. thesis, Michigan State University, Computer Science Department, 1993.

[9] www.ridgesoft.com last access in 25/12/2006.

[10] Andy Lindsay. "Robotics with the Boe-Bot".Version 2.2 .ISBN 1-928982-03-4
Copyright 2003 – 2004 by parallax Inc.

[11] Saeed B. Niku. "Introduction to Robotics Analysis, System, Applications".2001 Pearson education, Inc. Pearson Prentice Hall upper saddle river, New Jersey 07458

[12] John J. Craig. "Introduction to ROBOTICS Mechanics AND Control", 2005 Pearson education, Inc. Pearson Prentice Hall upper saddle river, NJ 07458

[13] J-C Latombe " Robot Motion Planning", IEEE Control S stems, vol. 15, n. 6, pp. 20-36, Dec. 1995.

[14] J S. Wit; “Vector Pursuit Path Tracking for Autonomous Ground vehicles”, PhD Thesis, University of Florida, 2000.

[15] E.S Tzafetas, J.D. Mathieu, and S. Tazafetas, "Overview of Experimental Evaluation of Potential Field-Based Robot Navigation" Advanced Robotic beyond 2000: The 29th International Symposium on Robotics, N.E.C, Birmingham, UK, 1998

[16] P. Webb and J.I. Robson, "An Intuitive Control and Sensory System for the Enhancement of Remotely Controlled Vehicle Operation", Proc Mechatronics 1998.

[17] R.A Brooks, "A robust layered control system for mobile robot", IEEE J. of Robotics and Automation, RA-2, PP .14-23, April 1986.

[18] Ronald C. Arkin, "Motor Scheme Based Navigation for Mobile Robot", proceeding of the IEEE Conference on Robotics and Automation , PP 264-D271, 1987.

[19] Jonathan H. Connel, "Navigation by Path Remembering", Proceeding of the 1988 SPIE Conference on Mobile Robots, PP 383-390.

[20] Andy Lindsay. "Advanced Robotics with the Toddler". Version 1.3. ISBN 1-928982-14-X. Copyright 2002-2004 by Parallax Inc.

7. APPENDICES

APPENDIXE A: The programs of obstacle avoidance and path tracking of the Boe-Bot robot.

1- Roaming with whiskers

```
                          '----- [ Title ]--------------------------
----------------------------------
' Robotics with the Boe-Bot - RoamingWithWhiskers.bs2
' Boe-Bot uses whiskers to detect objects, and navigates around them.
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
DEBUG "Program Running!"
' -----[ Variables ]-------------------------------------------------
--------
pulseCount VAR Byte ' FOR...NEXT loop counter.
' -----[ Initialization ]--------------------------------------------
--------
FREQOUT 4, 2000, 3000 ' Signal program start/reset.
```

```
' -----[ Main Routine ]-------------------------------------------------
--------
DO
IF (IN5 = 0) AND (IN7 = 0) THEN ' Both whiskers detect obstacle
GOSUB Back_Up ' Back up & U-turn (left twice)
GOSUB Turn_Left
GOSUB Turn_Left
ELSEIF (IN5 = 0) THEN ' Left whisker contacts
GOSUB Back_Up ' Back up & turn right
GOSUB Turn_Right
ELSEIF (IN7 = 0) THEN ' Right whisker contacts
GOSUB Back_Up ' Back up & turn left
GOSUB Turn_Left
ELSE ' Both whiskers 1, no contacts
GOSUB Forward_Pulse ' Apply a forward pulse
ENDIF ' and check again
LOOP
' -----[ Subroutines ]---------------------------------------------------
--------
Forward_Pulse: ' Send a single forward pulse.
PULSOUT 13,850
PULSOUT 12,650
PAUSE 20
RETURN
Turn_Left: ' Left turn, about 90-degrees.
FOR pulseCount = 0 TO 20
PULSOUT 13, 650
PULSOUT 12, 650
PAUSE 20
NEXT
RETURN
Turn_Right:
FOR pulseCount = 0 TO 20 ' Right turn, about 90-degrees.
PULSOUT 13, 850
PULSOUT 12, 850
PAUSE 20
NEXT
RETURN
Back_Up: ' Back up.
FOR pulseCount = 0 TO 40
PULSOUT 13, 650
PULSOUT 12, 850
PAUSE 20
NEXT
RETURN
```

2- Roaming with photoresistor sensor

```
' -----[ Title ]-------------------------------------------------------
--------
' Robotics with the Boe-Bot - RoamingWithPhotoresistorDividers.bs2
' Boe-Bot detects shadows photoresistors voltage divider circuit and
turns
' away from them.
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
DEBUG "Program Running!"
```

```
' -----[ Variables ]-------------------------------------------------
--------
pulseCount VAR Byte ' FOR...NEXT loop counter.
' -----[ Initialization ]---------------------------------------------
--------
FREQOUT 4, 2000, 3000 ' Start/restart signal.
' -----[ Main Routine ]-----------------------------------------------
--------
DO
IF (IN6 = 0) AND (IN3 = 0) THEN ' Both photoresistors detects
GOSUB Back_Up ' shadow, back up & U-turn
GOSUB Turn_Left ' (left twice).
GOSUB Turn_Left
ELSEIF (IN6 = 0) THEN ' Left photoresistor detects
GOSUB Back_Up ' shadow, back up & turn right.
GOSUB Turn_Right
ELSEIF (IN3 = 0) THEN ' Right photoresistor detects
GOSUB Back_Up ' shadow, back up & turn left.
GOSUB Turn_Left
ELSE ' Neither photoresistor detects
GOSUB Forward_Pulse ' shadow, apply a forward pulse.
ENDIF
LOOP
' -----[ Subroutines ]------------------------------------------------
--------
Forward_Pulse: ' Send a single forward pulse.
PULSOUT 12,650
PULSOUT 13,850
PAUSE 20
RETURN
Turn_Left: ' Left turn, about 90-degrees.
FOR pulseCount = 0 TO 20
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 20
NEXT
RETURN
Turn_Right:
FOR pulseCount = 0 TO 20 ' Right turn, about 90-degrees.
PULSOUT 12, 850
PULSOUT 13, 850
PAUSE 20
NEXT
RETURN
Back_Up: ' Back up.
FOR pulseCount = 0 TO 40
PULSOUT 12, 850
PULSOUT 13, 650
PAUSE 20
NEXT
RETURN
```

3- Roaming with infrared sensor

```
' -----[ Title ]-----------------------------------------------------
--------
' Robotics with the Boe-Bot - RoamingWithIr.bs2
' Adapt RoamingWithWhiskers.bs2 for use with IR pairs.
```

```
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.

DEBUG "Program Running!"
' -----[ Variables ]-------------------------------------------------
--------
irDetectLeft VAR Bit
irDetectRight VAR Bit
pulseCount VAR Byte
' -----[ Initialization ]--------------------------------------------
--------
FREQOUT 4, 2000, 3000 ' Signal program start/reset.
' -----[ Main Routine ]----------------------------------------------
--------
DO
FREQOUT 8, 1, 38500 ' Store IR detection values in
irDetectLeft = IN9 ' bit variables.
FREQOUT 2, 1, 38500
irDetectRight = IN0
IF (irDetectLeft = 0) AND (irDetectRight = 0) THEN
GOSUB Back_Up ' Both IR pairs detect obstacle
GOSUB Turn_Left ' Back up & U-turn (left twice)
GOSUB Turn_Left
ELSEIF (irDetectLeft = 0) THEN ' Left IR pair detects
GOSUB Back_Up ' Back up & turn right
GOSUB Turn_Right
ELSEIF (irDetectRight = 0) THEN ' Right IR pair detects
GOSUB Back_Up ' Back up & turn left
GOSUB Turn_Left
ELSE ' Both IR pairs 1, no detects
GOSUB Forward_Pulse ' Apply a forward pulse
ENDIF ' and check again
LOOP
' -----[ Subroutines ]-----------------------------------------------
--------
Forward_Pulse: ' Send a single forward pulse.
PULSOUT 13,850
PULSOUT 12,650
PAUSE 20
RETURN
Turn_Left: ' Left turn, about 90-degrees.
FOR pulseCount = 0 TO 20
PULSOUT 13, 650
PULSOUT 12, 650
PAUSE 20
NEXT
RETURN
Turn_Right:
FOR pulseCount = 0 TO 20 ' Right turn, about 90-degrees.
PULSOUT 13, 850
PULSOUT 12, 850
PAUSE 20
NEXT
RETURN
Back_Up: ' Back up.
FOR pulseCount = 0 TO 40
PULSOUT 13, 650
PULSOUT 12, 850
PAUSE 20
NEXT
```

```
RETURN
```

4- Test frequency sweep

```
' -----[ Title ]-------------------------------------------------------
--------
' Robotics with the Boe-Bot - TestLeftFrequencySweep.bs2
' Test IR detector distance responses to frequency sweep.
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
' -----[ Variables ]----------------------------------------------------
--------
freqSelect VAR Nib
irFrequency VAR Word
irDetect VAR Bit
distance VAR Nib
' -----[ Initialization ]-----------------------------------------------
--------
DEBUG CLS,
" OBJECT", CR,
"FREQUENCY DETECTED", CR,
"--------- --------"
' -----[ Main Routine ]-------------------------------------------------
--------
DO
distance = 0
FOR freqSelect = 0 TO 4
LOOKUP freqSelect,[37500,38250,39500,40500,41500], irFrequency
FREQOUT 8,1, irFrequency
irDetect = IN9
distance = distance + irDetect
DEBUG CRSRXY, 4, (freqSelect + 3), DEC5 irFrequency
DEBUG CRSRXY, 11, freqSelect + 3
IF (irDetect = 0) THEN DEBUG "Yes" ELSE DEBUG "No "
PAUSE 100
NEXT
DEBUG CR,
"--------- --------", CR,
"Zone ", DEC1 distance
LOOP
```

5- How to measure RC decay time

```
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
timeLeft VAR Word
DO
HIGH 3
PAUSE 2
RCTIME 3,1,timeLeft
DEBUG HOME, "timeLeft = ", DEC5 timeLeft
PAUSE 1000
HIGH 6
```

```
PAUSE 2
RCTIME 6,1,timeLeft
DEBUG HOME, "                    timeright = ", DEC5 timeLeft
PAUSE 100
LOOP
```

6- FollowingBoeBot.bs2 repeats the proportional control loop

```
  ' -----[ Title ]-----------------------------------------------------
----------
' Robotics with the Boe-Bot - FollowingBoeBot.bs2
' Boe-Bot adjusts its position to keep objects it detects in zone 2.
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
DEBUG "Program Running!"
' -----[ Constants ]--------------------------------------------------
--------
Kpl CON -35
Kpr CON 35
SetPoint CON 2
CenterPulse CON 750
' -----[ Variables ]--------------------------------------------------
--------
freqSelect VAR Nib
irFrequency VAR Word
irDetectLeft VAR Bit
irDetectRight VAR Bit
distanceLeft VAR Nib
distanceRight VAR Nib
pulseLeft VAR Word
pulseRight VAR Word
' -----[ Initialization ]---------------------------------------------
--------
FREQOUT 4, 2000, 3000
' -----[ Main Routine ]-----------------------------------------------
--------
DO
GOSUB Get_Ir_Distances
' Calculate proportional output.
pulseLeft = SetPoint - distanceLeft * Kpl + CenterPulse
pulseRight = SetPoint - distanceRight * Kpr + CenterPulse
GOSUB Send_Pulse
LOOP
' -----[ Subroutine - Get IR Distances ]------------------------------
--------
Get_Ir_Distances:
distanceLeft = 0
distanceRight = 0
FOR freqSelect = 0 TO 4
LOOKUP freqSelect,[37500,38250,39500,40500,41500], irFrequency
FREQOUT 8,1,irFrequency
irDetectLeft = IN9
distanceLeft = distanceLeft + irDetectLeft
FREQOUT 2,1,irFrequency
irDetectRight = IN0
distanceRight = distanceRight + irDetectRight
NEXT
```

```
RETURN
' -----[ Subroutine - Get Pulse ]-----------------------------------
--------
Send_Pulse:
PULSOUT 13,pulseLeft
PULSOUT 12,pulseRight
PAUSE 5
RETURN
```

7- Following Strip

```
' -----[ Title ]----------------------------------------------------
--------
' Robotics with the Boe-Bot - StripeFollowingBoeBot.bs2
' Boe-Bot adjusts its position to move toward objects that are closer
than
' zone 3 and away from objects further than zone 3. Useful for
following a
' 2.25 inch wide vinyl electrical tape stripe.
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
DEBUG "Program Running!"
' -----[ Constants ]-------------------------------------------------
--------
Kpl CON 35 ' Change from -35 to 35
Kpr CON -35 ' Change from 35 to -35
SetPoint CON 3 ' Change from 2 to 3.
CenterPulse CON 750
' -----[ Variables ]-------------------------------------------------
--------
freqSelect VAR Nib
irFrequency VAR Word

irDetectLeft VAR Bit
irDetectRight VAR Bit
distanceLeft VAR Nib
distanceRight VAR Nib
pulseLeft VAR Word
pulseRight VAR Word
' -----[ Initialization ]--------------------------------------------
--------
FREQOUT 4, 2000, 3000
' -----[ Main Routine ]----------------------------------------------
--------
DO
GOSUB Get_Ir_Distances
' Calculate proportional output.
pulseLeft = SetPoint - distanceLeft * Kpl + CenterPulse
pulseRight = SetPoint - distanceRight * Kpr + CenterPulse
GOSUB Send_Pulse
LOOP
' -----[ Subroutine - Get IR Distances ]------------------------------
--------
Get_Ir_Distances:
distanceLeft = 0
distanceRight = 0
```

```
FOR freqSelect = 0 TO 4
LOOKUP freqSelect,[37500,38250,39500,40500,41500], irFrequency
FREQOUT 8,1,irFrequency
irDetectLeft = IN9
distanceLeft = distanceLeft + irDetectLeft
FREQOUT 2,1,irFrequency
irDetectRight = IN0
distanceRight = distanceRight + irDetectRight
NEXT
RETURN
' -----[ Subroutine - Get Pulse ]-------------------------------------
--------
Send_Pulse:
PULSOUT 13,pulseLeft
PULSOUT 12,pulseRight
PAUSE 5
RETURN
```

8- Navigation Boe-Bot robot (obstacle avoidance)

```
' {$STAMP BS2}
' {$PBASIC 2.5}
X_axis VAR Word
y_axis VAR Word
 distance VAR Word
degr VAR Word
pulsCount VAR Byte
anglemod VAR Word
counter VAR Word
sine VAR Word
d VAR Word
pulseCount VAR Byte
time VAR Word
newdist VAR Word
direct3 VAR Word
distanceservo   VAR Byte
distanceservo2  VAR Byte

   Ping            PIN     15

main:
'DEBUG "enter value of x axis           "
 x_axis=80
'DEBUG "enter value of y axis           "
 y_axis=80

distance=SQR((x_axis*x_axis)+(y_axis*y_axis))

DEBUG "  distance    =",DEC distance

DEBUG distance,"."


sine=(x_axis*100)/distance

   DEBUG "  sine    =",DEC sine
```

```
 IF (99<sine)AND(sine=<100) THEN
d=20

 GOSUB turn


  ELSEIF (98<sine)AND(sine<=99) THEN
  d=19
  GOSUB turn


  ELSEIF (64<sine)AND(sine<=70) THEN
  d=10
  GOSUB turn


 ELSEIF (96<sine)AND(sine<=98) THEN
  d=18
  GOSUB turn


    ELSEIF (93<sine)AND(sine<=96) THEN
  d=17
  GOSUB turn


   ELSEIF (90<sine)AND(sine<=93) THEN
  d=15
  GOSUB turn


   ELSEIF (86<sine)AND(sine<=90) THEN
  d=14
  GOSUB turn


   ELSEIF (81<sine)AND(sine<=86) THEN
  d=13
  GOSUB turn


   ELSEIF (76<sine)AND(sine<=81) THEN
  d=12
  GOSUB turn


   ELSEIF (70<sine)AND(sine<=76) THEN
  d=11
  GOSUB turn


   ELSEIF (57<sine)AND(sine<=64) THEN
  d=9
  GOSUB turn


   ELSEIF (50<sine)AND(sine<=57) THEN
  d=8
  GOSUB turn
```

```
       ELSEIF (42<sine)AND(sine<=50) THEN
      d=6
      GOSUB turn


       ELSEIF (34<sine)AND(sine<=42) THEN
      d=5
      GOSUB turn


       ELSEIF (25<sine)AND(sine<=34) THEN
      d=4
      GOSUB turn


       ELSEIF (17<sine)AND(sine<=25) THEN
      d=3
      GOSUB turn


       ELSEIF (sine<=17) THEN
      d=1
      GOSUB turn
        ENDIF

   time=0
loop1:
   distance=distance-(newdist+20)
DO  WHILE (time<distance)
GOSUB no
   GOSUB sound
       LOOP


     END

 turn:
 FOR pulscount=0 TO d
 PULSOUT 12,850
 PULSOUT 13,850
 PAUSE 20
 NEXT

 no:
    FOR pulscount=0 TO 0
 PULSOUT 12,750
 PULSOUT 13,750
 PAUSE 200
 NEXT

 sound:
  PULSOUT 15, 5
PULSIN 15, 1, time
DEBUG HOME, "time = ", DEC5 time
time = time ** 2251


DEBUG CR, "Distance = ", DEC4 time, " cm"
PAUSE 100
```

```
    newdist=time-10

    d=newdist*(22)
     direct3=distance-(newdist+20 )
     DEBUG CR, "Distance = ", DEC4 newdist, " cm"
   IF (distance<time) THEN
GOSUB goal
ELSE

GOSUB forward1
GOSUB chek

ENDIF

goal:
   FOR pulscount=0 TO (distance*22)/10
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT
     END

  forward1:
 FOR pulscount=0 TO newdist*2
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT

 chek:
 FOR pulseCount = 0 TO 20              ' Number Of Pulses To Spin
    LOW Ping                           ' Force PING))) Line Low
    PULSOUT 14, 850              ' Ping Servo 45 Left Pulse Value
    PULSOUT Ping, 5                    ' Activate PING)))
    PULSIN  Ping, 1, distanceservo           ' Receive Distance Value
    PAUSE 20
                                ' Refresh Delay
  NEXT



'
******************************************************************
***
' * USE THE APPROPRIATE PULSOUT VALUE TO MAKE YOUR PING)))
*
' * TURN 45 DEGREES RIGHT.
*
'
******************************************************************
***

  FOR pulseCount = 0 TO 20             ' Number Of Pulses To Spin
    LOW Ping                           ' Force PING))) Line Low
    PULSOUT 14, 650              ' Ping Servo 45 Right Pulse Value
    PULSOUT Ping, 5                    ' Activate PING)))
    PULSIN  Ping, 1, distanceservo2          ' Receive Distance Value
```

```
    PAUSE 20
                                ' Refresh Delay
  NEXT

   FOR pulseCount = 0 TO 20              ' Number Of Pulses To Spin
    LOW Ping                            ' Force PING))) Line Low
    PULSOUT 14, 750              ' Ping Servo 45 Right Pulse Value

    PAUSE 20
                                ' Refresh Delay
      NEXT
   IF (distanceservo<distanceservo2) THEN
      GOSUB turnrl
     GOSUB FORward10l
     GOSUB turnll
     GOSUB FORward20l
     GOSUB turnl2l
     GOSUB forward30l
     GOSUB turnr2l



     ELSEIF (distanceservo>distanceservo2) THEN
      GOSUB turnr
     GOSUB FORward10
     GOSUB turnl
     GOSUB FORward20
     GOSUB turnl2
     GOSUB forward30
     GOSUB turnr2

        ENDIF
      END

 turnr:
FOR pulscount=0 TO 20
 PULSOUT 13,850
 PULSOUT 12,850
 PAUSE 20
 NEXT

 forward10:
 FOR pulscount=0 TO 60
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT

 turnl:
FOR pulscount=0 TO 20
 PULSOUT 13,650
 PULSOUT 12,650
 PAUSE 20
 NEXT

 forward20:
 FOR pulscount=0 TO 80
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
```

```
 NEXT
 turnl2:
FOR pulscount=0 TO 20
 PULSOUT 13,650
 PULSOUT 12,650
 PAUSE 20
 NEXT
 forward30:
 FOR pulscount=0 TO 60
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT
 turnr2:
FOR pulscount=0 TO 20
 PULSOUT 13,850
 PULSOUT 12,850
 PAUSE 20
 NEXT
  GOTO loop1
  END



''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''
  turnrl:
FOR pulscount=0 TO 20
 PULSOUT 13,650
 PULSOUT 12,650
 PAUSE 20
 NEXT

 forward10l:
 FOR pulscount=0 TO 60
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT

 turnll:
FOR pulscount=0 TO 20
 PULSOUT 13,850
 PULSOUT 12,850
 PAUSE 20
 NEXT

 forward20l:
 FOR pulscount=0 TO 80
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT
 turnl2l:
FOR pulscount=0 TO 20
 PULSOUT 13,850
 PULSOUT 12,850
 PAUSE 20
 NEXT
```

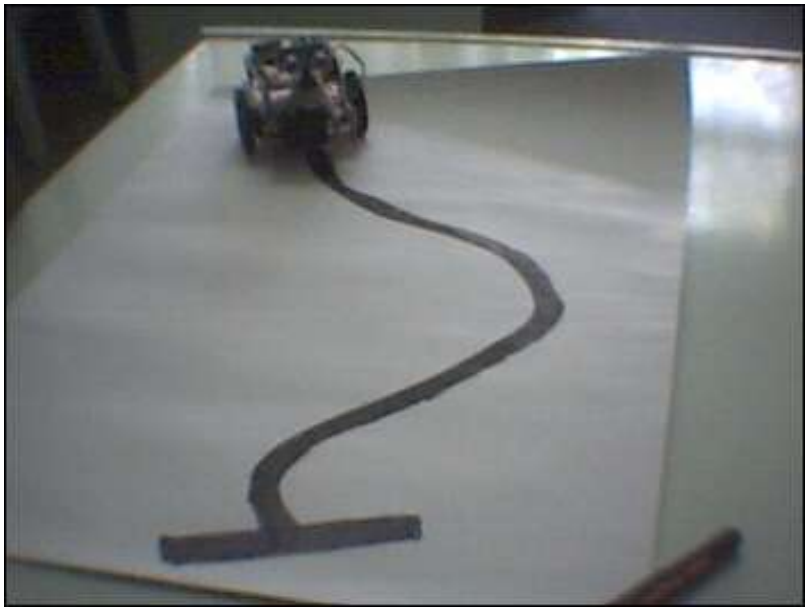
```
 forward30l:
 FOR pulscount=0 TO 60
 PULSOUT 13,850
 PULSOUT 12,650
 PAUSE 20
 NEXT
 turnr2l:
FOR pulscount=0 TO 20
 PULSOUT 13,650
 PULSOUT 12,650
 PAUSE 20
 NEXT
 GOTO loop1
 END
```

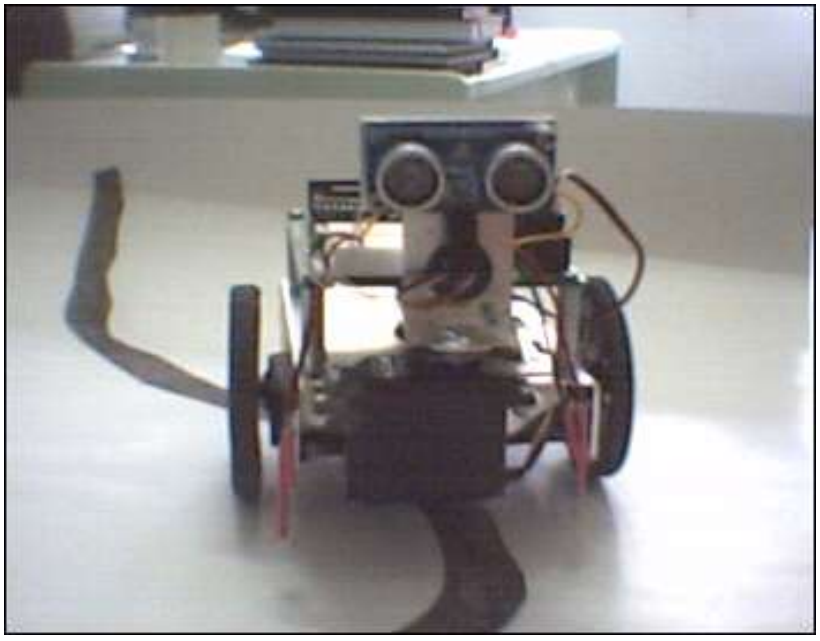
9- Path tracking Boe-Bot program

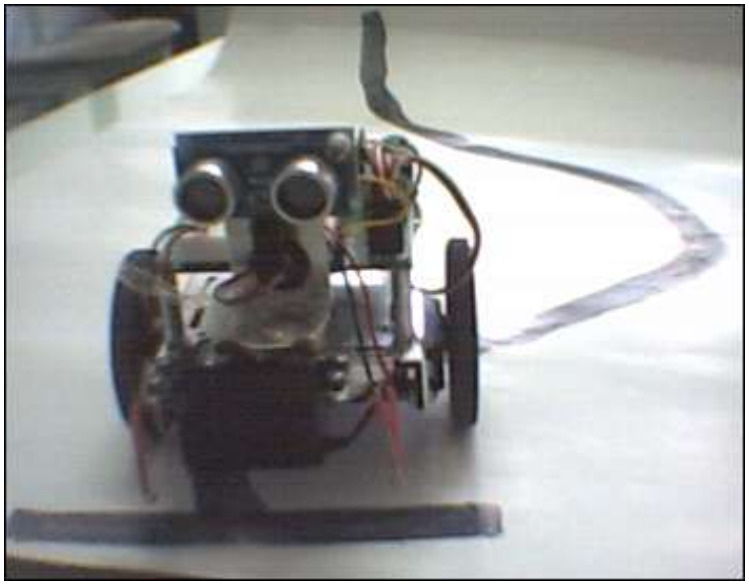
```
' {$STAMP BS2}
' {$PBASIC 2.5}
timeLeft VAR Word
timeright VAR Word
 pulscount VAR Byte
  DO
GOSUB Test_Photoresistors
GOSUB Navigate
LOOP
' -----[ Subroutine - Test_Photoresistors ]--------------------------
--------
Test_Photoresistors:
HIGH 6 ' Left RC time measurement.
PAUSE 3
RCTIME 6,1,timeLeft
HIGH 3 ' Right RC time measurement.
PAUSE 3
RCTIME 3,1,timeRight
RETURN
' -----[ Subroutine - Navigate ]-------------------------------------
--------
Navigate:
DO
IF (timeLeft <48) AND (timeRight <48) THEN
PULSOUT 13, 850 ' Both detect wight,
PULSOUT 12, 650 ' full speed forward.
ELSEIF (timeLeft >48) AND (timeRight <48) THEN
FOR pulscount = 0 TO 1
PULSOUT 13, 650 ' left sensor detects black
PULSOUT 12, 650 ' rotate to left
NEXT
RETURN
  ELSEIF (timeLeft <48) AND (timeRight >48) THEN
FOR pulscount = 0 TO 1
PULSOUT 13, 850 ' right sensor detect black
PULSOUT 12, 850 ' rotate to right
NEXT
RETURN
ELSEIF (timeLeft >48) AND (timeRight >48) THEN
FOR pulscount = 0 TO 1
PULSOUT 13, 750 ' both detect black
```

```
PULSOUT 12, 750 ' get the goal (stop)
NEXT
RETURN
 ENDIF
PAUSE 20 ' Pause between pulses.
RETURN
LOOP
```

APPENDIX B: Actual images of the practical work of the path tracking algorithm.







APPENDIX C: Actual images for the practical results of obstacle avoidance algorithm.