

**GRID FOLLOWING ROBOT USING EVENT-DRIVEN PROGRAMMING
TECHNIQUE AND LED-LDR SENSORS**

By

NURUL ZAHIDAH BT MD. HASIDIN

FINAL DISSERTATION

*Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)*

SEPTEMBER 2011

Universiti Teknologi Petronas

Bandar Seri Iskandar

31750 Tronoh

Perak Darul Ridzuan

© Copyright 2011

by

Nurul Zahidah Bt Md. Hasidin, 2011

CERTIFICATION OF APPROVAL

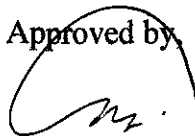
GRID FOLLOWING ROBOT USING EVENT-DRIVEN PROGRAMMING TECHNIQUE AND LED-LDR SENSORS

by

Nurul Zahidah Bt Md. Hasidin

A project interim report submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved by



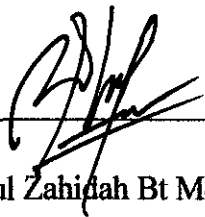
Mr Abu Bakar Sayuti Bin Hj Mohamad Saman
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

SEPTEMBER 2011

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.



Nurul Zahidah Bt Md. Hasidin

ABSTRACT

Grid Trekker is a grid follower robot which is a type of mobile robot that uses line detection for its navigation. The ultimate aim of this project is to build a mobile robot that can maneuver correctly through paths determined by its user. The mobile robot must be able to maneuver based on grids marked on the floor. Using Parallax Board of Education robotics kit, this project focuses on the sensor assembly using combination of LED and light dependant resistor (LDR) which is simple but effective. Another aim of this project is the program development using event-driven programming. Unlike traditional programming, where the control flow is determined by the program structure, the control flow of event driven programs is largely driven by external events. The main routine basically reacts to events based on its current state and transition to the next state accordingly, much like a Finite State Machine (FSM). Implemented in PBASIC, this technique was successfully implemented in programming the Grid Trekker.

ACKNOWLEDGEMENTS

I would like to express my many thanks to my supervisor of this project, Mr Abu Bakar Sayuti Bin Hj Mohamad Saman for the motivation and guidance in this project. He has constantly provided valuable guidance and advice which has motivated me to contribute my fullest to this project. Step by step guidance motivates me to keep moving and make this project done. Not forgetting my family and friends who has giving tremendous support and concern along the way which greatly motivated me to keep putting effort on completing this project. I would like to thank the Final Year Project Committees to have organized and assisting us for these two semesters on completing this project. Finally, I would like to thank Universiti Teknologi PETRONAS (UTP) for providing us with the excellent environment and access to facilities to complete this project on our own initiative.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS.....	v
LIST OF FIGURES	vii
LIST OF TABLES.....	ix
CHAPTER 1	1
INTRODUCTION	1
1.1. Background Study.....	1
1.2. Problem Statement	2
1.3. Objective	2
1.4. Scope of Study	2
CHAPTER 2	4
LITERATURE REVIEW AND THEORY.....	4
2.1. Mobile robot.....	4
2.1.1. Microcontroller	4
2.1.2. Servo motor.....	4
2.1.3. Line Sensor	5
2.1.3.1. Light Dependant Resistor (LDR) & LED	5
2.1.3.2. How the LDR works	6
2.2. Line follower robot & grid follower robot.....	7
2.3. Finite State Machine (FSM).....	7
2.3.1. State Transition Matrix	8
2.3.2. State Transition Diagram	9
2.4. Pseudo-code	11
2.5. Conventional programming	11
2.6. Event-driven programming	12
2.6.1. C and PBASIC Program	12

CHAPTER 3 METHODOLOGY	14
3.1. Research Methodology	14
3.1.1. Building the line sensor using Light Dependant Resistors (LDR) and white LEDs.....	14
3.1.2. Testing Sensor Response to light reflection from LED	15
3.1.2. Grid Following Behavior Modeling.....	17
3.2. Key milestone	18
3.3. Project activities.....	19
3.4. Tools used	20
3.4.1. Hardware.....	20
3.4.2. Software	21
3.4.3. Computer system requirement:	21
CHAPTER 4 RESULT AND DISCUSSION	22
4.1. Hardware assembly	22
4.1.1. Servo Motors.....	23
4.1.2. Line sensor construction	23
4.1.2.1. Sensor drawbacks.....	27
4.1.2.1.1. Sensor Position.....	27
4.1.2.1.2. LEDs as Indicators to Verify Input Detection.....	28
4.1.2.1.3. Number of Sensors.....	28
4.2. Building the Grid Track	29
4.3. Algorithm Development	29
4.3.1. Line Following.....	30
4.3.2. Grid Following.....	36
CHAPTER 5 CONCLUSION.....	49
REFERENCES	50

LIST OF FIGURES

FIGURE 1: SCHEMATIC DIAGRAM FOR LDR SENSOR CIRCUIT [].....	6
FIGURE 2: FSM CONCEPT FLOW	7
FIGURE 3: STATE TRANSITION DIAGRAM	9
FIGURE 4: STATE TRANSITION DIAGRAM FOR SIMPLE LINE FOLLOWING BEHAVIOR..	10
FIGURE 5: BASIC STAMP EDITOR V2.5.1	13
FIGURE 6: BOTH COUPLE OF LED-LDR SENSOR DETECTS LIGHT	16
FIGURE 7: LEFT SENSOR DOES NOT DETECT LIGHT ON A DARK SURFACE	16
FIGURE 8: TESTING SENSOR RESPONSE.....	17
FIGURE 9: KEY MILESTONE OF THE PROJECT	18
FIGURE 10: ACTIVITY FLOW	19
FIGURE 13: 6V POWER SOURCE.....	22
FIGURE 11: PARALLAX BOE-BOT.....	22
FIGURE 12: PINS AND SERVO ASSIGNMENT	22
FIGURE 15: SERVO MOTOR.....	23
FIGURE 14: 3 PINS PLUG SERVO MOTOR.....	23
FIGURE 16: LED SENSOR CIRCUIT DIAGRAM [4].....	23
FIGURE 17: LDR SENSOR CIRCUIT DIAGRAM [4].....	23
FIGURE 18: (A) SIDE VIEW OF THE LINE SENSOR,	
(B) FRONT VIEW OF THE LINE SENSOR	24
FIGURE 19: LINE SENSOR CIRCUIT CONSISTING OF LED-LDR	25
FIGURE 20: VEROBOARD LINE SENSOR CIRCUIT	26
FIGURE 21: SENSOR ASSIGNED TO I/O PIN3 AND PIN6, POWER SOURCE CONNECTED ...	26
FIGURE 22: SENSOR IS MOUNTED UNDER THE GRID FOLLOWER ROBOT	27
FIGURE 23: POSITION OF SENSOR AFFECTS ACCURACY OF TURN AT JUNCTION	28
FIGURE 24: LED INDICATORS ADDED	28
FIGURE 25: POSSIBLE POSITION FOR THE THIRD SENSOR;	
(A) AT THE FRONT, (B) AT THE BACK, (C) BETWEEN THE EXISTING TWO	29
FIGURE 26: GRID FOLLOWER ROBOT MANEUVER ACCORDING TO GRIDS	29
FIGURE 27: LINE FOLLOWING ROBOT BEHAVIOR.....	30
FIGURE 28: STATE TRANSITION DIAGRAM FOR SIMPLE LINE FOLLOWING	
BEHAVIOR FROM CHAPTER 2	31
FIGURE 29: MOBILE ROBOT ABLE TO MANEUVER ON GRIDS ACCORDING TO THE	
FORMATION SET	36

FIGURE 30: GRID TREKKING BEHAVIOR.....	37
FIGURE 31: FSM STATE CHART FOR GRID FOLLOWER ROBOT.....	40
FIGURE 32: LEDS INDICATORS SUBROUTINES ARE CALLED IN THE MAIN ROUTINE. REFER FULL PROGRAM AT APPENDIX C-B(III)	47
FIGURE 33: SENSOR INDICATOR AND JUNCTION COUNTER LED INDICATORS SUBROUTINES. REFER FULL PROGRAM AT REFER FULL PROGRAM AT APPENDIX C-B(III)	48

LIST OF TABLES

TABLE 1: TRANSITION MATRIX FOR SIMPLE LINE TREKKER BEHAVIOR.....	9
TABLE 2: BASIC MOVEMENTS OF THE LINE FOLLOWING APPLICATION	15
TABLE 3: HARDWARE USED BASED ON THE PARALLAX BOE-BOT STUDENT GUIDE [8]21	
TABLE 4: COMPONENTS FOR LINE SENSOR.....	24
TABLE 5: LINE FOLLOWING BEHAVIOR OVERVIEW	30
TABLE 6: TRANSITION MATRIX FOR SIMPLE LINE TREKKER BEHAVIOR (FROM CHAPTER 2)	31
TABLE 7: GRID TREKKING BEHAVIOR	38
TABLE 9: ARRAYS.....	39
TABLE 8: STATE TRANSITION MATRIX FOR GRID TREKKING	39

CHAPTER 1

INTRODUCTION

1.1. Background Study

A robot is a re-programmable, multifunctional device designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks [1]. In practice, it is usually an electro-mechanical machine which is guided by computer or electronic programming, and is thus able to do tasks on its own. Robots can be classified to several classes by the Japanese Industrial Robot Association (JIRA) that are manual handling device, fixed sequence robot, variable sequence robot, playback robot, numerical control robot and intelligent robot [1]. However, generally robots can be classified to two types; fixed position robot and mobile robot.

Mobile robots have the capability to move around in their environment and are not fixed to one physical location [1]. Hence, the ability to move autonomously in the environment opened a wide scope of its application such as task that involves transportation, exploration, surveillance, guidance, etc. Autonomous mobile robots offer a great medium of testing on intelligent behavior. In order to autonomously move, mobile robots need sensing, interpretation, cognition and coordination as the input to be processed. The type of sensor equipped to it determines the way the robot navigates. A grid follower robot uses line detection for navigation.

A grid follower robot is a mobile robot that can detect and follow a line drawn on a floor [2]. Generally, the path is predefined and can be either visible like a black line on a white surface with a high contrasted color or it can be invisible like a magnetic field [2]. This line may be as simple as a physical white line on the floor or as complex lines e.g. embedded lines, magnetic lines and laser guide lines. In order to detect these specific lines, various sensing schemes can be employed [3]. It tracks

the path based on difference of colors since only the bright part reflects back the lights to the sensor. To make it autonomously move based on the path drawn, the robot must be equipped with suitable sensors [3].

Thus, because of its mechanism and the way it navigates, this robot is named the grid follower robot. Other names of the grid follower robot is line follower and it can be used in various areas such as the industrial automated equipment carriers, automated cars, tour guides in museums and other similar applications. The mechanism is described further in chapter 2.

1.2. Problem Statement

A grid follower robot must have the ability to recognize grids on the floor by means of a line sensor assembly consisting Light Dependant Resistor (LDR) and white LED. The sensor detects the line as input. The input will be processed by the algorithm to perform several required tasks such as detecting a line, moving forward, backward, reversing, taking various degrees of turns and stopping. Most importantly the robot requires an algorithm which gives the machine an artificial intelligence such as the ability to maneuver based on how it is programmed.

1.3. Objective

The ultimate aim of this project is to build a mobile robot that can maneuver correctly through paths determined by its user. The mobile robot must be able to maneuver based on grids marked on the floor. Using a ready Parallax Board of Education robotics kit, this project focuses on the sensor assembly and the program development using Finite State Machine model.

1.4. Scope of Study

The grid follower robot covers the area of digital electronics, microprocessor, microcontroller, and programming.

The project will evolve around the program development using event-driven programming. Knowledge in programming will definitely be tested to ensure the program developed able to coordinate the microcontroller to the actuators (motors and wheels) of the grid follower robot. This will give the grid follower robot a sense of artificial intelligence which is the ability to move autonomously based on the pre-drawn grids. This involves programming the Parallax BASIC Stamp® 2 microcontroller using PBASIC Language. Event driven programming is used for more organized and reliable program.

The project will also cover the knowledge of circuit theory to fulfill the objective of the sensor development for the mobile robot. The cheapest but effective line sensor which consists of LEDs and Light Dependant Resistors (LDR) will be built.

CHAPTER 2

LITERATURE REVIEW AND THEORY

2.1. Mobile robot

Mobile robot consists of several important parts that are the microcontroller, sensor and the servo motor.

2.1.1. Microcontroller

A microcontroller is a programmable device that is designed into your digital gadgets [4]. Microcontroller for embedded systems is different with embedded microcontroller as the latter have its ROM burned with a purpose for specific functions needed by the system [5]. A microcontroller has a CPU (a microprocessor) in addition to a fixed amount of RAM, ROM, I/O ports and timer all on a single chip. This means no external memory, I/O, or timer can be added to it.

A microcontroller differs from microprocessor where microprocessor contains no RAM, no ROM, and no I/O ports on its chip. Microprocessors can have external RAM, ROM and I/O but this will make the system bulkier causing it to be less-favorable and not ideal for applications with limited cost and space [5].

2.1.2. Servo motor

Servo motor is a DC motor with more than two electrical terminals. Servo motors is the three-wire DC servo motor that is often used for a control surface on a model airplane or a steering motor on a radio-controlled car. There are two types of servo motor which are the 180° and 360°.

The original servo motor is the 180° which is an assembly of a DC gear head motor, a position sensor on the shaft and an integrated circuit for control. It rotates at

the range of 180° and this rotation range is not suitable for wheeled robot propulsions. This type of servo motor converts electrical to mechanical in which an electrical input determines the position of the armature of a motor [6].

For wheeled robot, the second type of servo motor which is the continuous servo 360° is used. This type of servo has been modified to have 360° rotation. The servo motors used for this robot will enable the robot to make various degrees of turns and movements such as turn, moving forward and backward.

2.1.3. Line Sensor

Autonomous mobile robots offer a great medium of testing on intelligent behavior. In order to autonomously move, mobile robots needs sensing, interpretation, cognition and coordination as the input to be processed. The type of sensor equipped to it determines the way the robot navigates. A grid follower robot uses line detection for navigation. This line sensor consists of photoresistors or also be called as Light Dependant Resistors (LDR) and white LEDs.

2.1.3.1. Light Dependant Resistor (LDR) & LED

Light dependant resistors or also known as photoresistors are simply variable resistors in many ways similar to potentiometers, except that the resistance change is caused by a change in light level rather than turning a knob. Photoresistors are easy to interface to a microcontroller [6].

2.1.3.2. How the LDR works

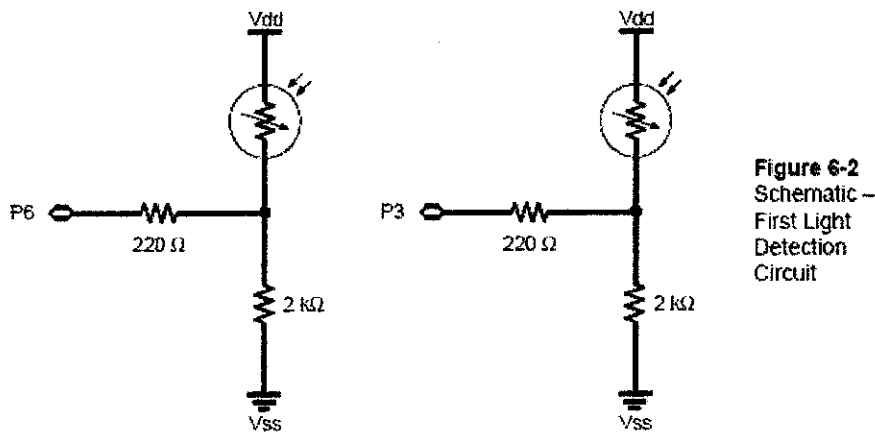


Figure 1: Schematic diagram for LDR sensor circuit [ERROR! BOOKMARK NOT DEFINED.]

In this project, BASIC Stamp module is used. The BASIC Stamp microcontroller I/O pin can function as an output or an input. As an output, the I/O pin can send a high (5 V) or low (0 V) signal [Error! Bookmark not defined.]. Up to this point, high and low signals have been used to turn LED circuits on and off thus it then controls the servos.

The BASIC Stamp I/O pin can also function as an input. As an input, the I/O pin does not apply any voltage to the circuit it is connected to. Instead, it just receives input without any actual effect on the circuit. The input registers will store values that indicated whether or not the LDR detects light. Different with early systems which needs constant voltage for the I/O pin, an I/O pin set to input does not necessarily need 5 V applied to it to make its input register store a 1. Anything above 1.4 V will make the input register for an I/O pin store a 1. Similarly, an I/O pin must not get 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 [4].

When a BASIC Stamp I/O pin is an input, the circuit behaves as though neither the I/O pin nor 220 Ω resistors are present. Based on Figure 1 above, the resistance, R of the LDR could be a few ohms if the light is very bright, or it could be in range

of 50 k Ω in complete darkness. In a room with high intensity of light, the resistance could be as small as a 1 k Ω or it can be as large as 25 k Ω in low intensity light [4].

When the resistance varies, the output voltage, V_o will also vary. Larger R produces small output voltage, V_o and vice versa. The output voltage, V_o is what the I/O pin is detecting when it is functioning as an input. If this circuit is connected to IN6, when the voltage at V_o is above 1.4 V, IN6 will store a 1 and vice versa [4].

2.2. Line follower robot & grid follower robot

There are some differences of the line follower robot and the grid follower robot. The line follower robot follows a simple single-line track while grid follower robot requires detection of complex gridlines and remembering the position on which it is at that time. This requires less number of states or movements of the robot. Grid follower robot have the ability to detect intersections and react based on program loaded to it; whether to turn at the intersection or not. This requires more states of movement, more complex and detailed behavior modeling using Finite State Machine concept.

2.3. Finite State Machine (FSM)

Finite state machine is a concept used to build a behavior control program by implementing the behaviors as finite-state machines. FSM is a computational element consisted of a collection of states. With the main objective of FSM to generate actions, consider the diagram Figure 2 showing a Control System controls and Application [7].

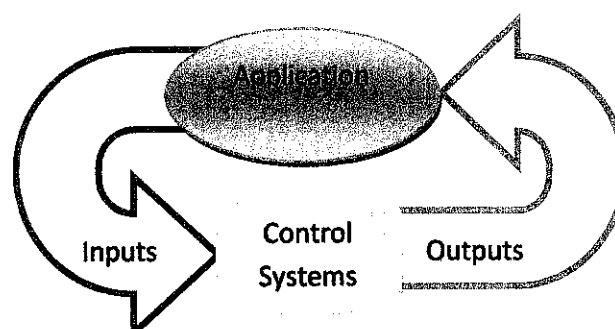


Figure 2: FSM concept flow [7]

The control system receives inputs from the application. Control system process and analyze it before output is produced to affect the application. A simple logical condition showing portray clear model on describing the control model above:

IF (input conditions) THEN Output

The logical condition above assigns the logical inputs to the actions (output) required to be done according to the application of the system [7]. The input conditions are written in logical expressions which are formulated from Boolean algebra. There may be more than one input feed, and this requires logical operators such as AND and OR. Input conditions can be written as below:

$(LDRDetectRight=0) \text{ AND } (LDRDetectLeft=0)$

$(LDRDetectRight=0) \text{ OR } (LDRDetectLeft=0)$

However, this simple model is sufficient only for trivial systems where for more realistic applications needs more complicated control model. Finite State Machine (FSM) is one of the most powerful models to describe behavior [7]. Behavior modeling is to describe what to do in all imaginable situations [7].

The Finite State Machine introduces concept of a states. All states represent all possible situations for the state machine. To be able execute a behavior or movement; there must be a kind of memory on how the state machine reach the present situation. As the machine runs, the states changes continuously, and the output depends on the current state and also the input [7].

To keep the machine understandable, there must be a limit to the number of states. Designer must take this into consideration and use only important and required states for simplification of the system.

Finite state machine tools are mainly the state transition diagram and the state transition table.

2.3.1. State Transition Matrix

A state transition table is a table represents states and the next state [7]. It displays the conditions that must be achieved for transition to occur [7]. For example refer to Table 1 for the transition matrix of a line following robot.

Present states\Next states	00	01	10	11
Idle	-	-	-	Forward
Forward (11)	Stop	Turn left	Turn right	Forward
Turn Left (01)	Stop	Turn left	Turn right	Forward
Turn Right (10)	Stop	Turn left	Turn right	Forward
Stop (00)	Stop	Turn left	Turn right	Forward

Table 1: Transition matrix for simple line trekker behavior

Referring to table 1, the line following robot have five basic states and by default in idle state. The line following robot will only start moving forward when detects input from its pair of sensors bits 11. In other states, the line following robot will stop, turn left, turn right and stop if the input feed is 00, 01, 10 and 00 respectively.

2.3.2. State Transition Diagram

A state transition diagram is a graphical representation equivalent to the state transition matrix [7]. It is similar to the transition matrix but easier to understand. It consists of two elements: a circle to denote states and an arc (or arrows) for the transition. The transition condition is written over the arc. Transition from one state to another will only happen if the transition condition is true.

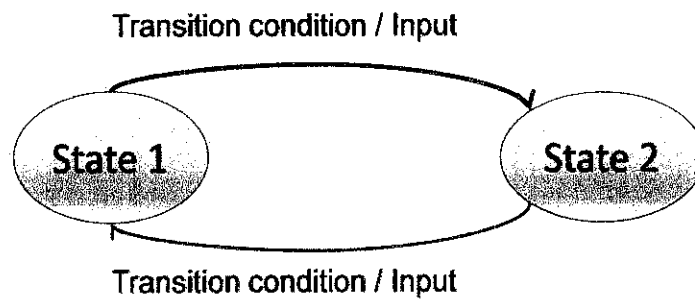


Figure 3: State transition diagram

Figure 4 below is an example of a simple line trekking robot state transition diagram. See that the transition condition is actually the input feed from its pair of sensors. Transition condition will trigger the state machine execution environment only to the current state and then it disappears. At idle state, when the line sensors feeds 11, 'Idle' state will be triggered to move to 'Forward' state only. Then, in 'Forward' state, any inputs (00, 01, 10 and 11) detected will trigger the current state to execute the next state accordingly.

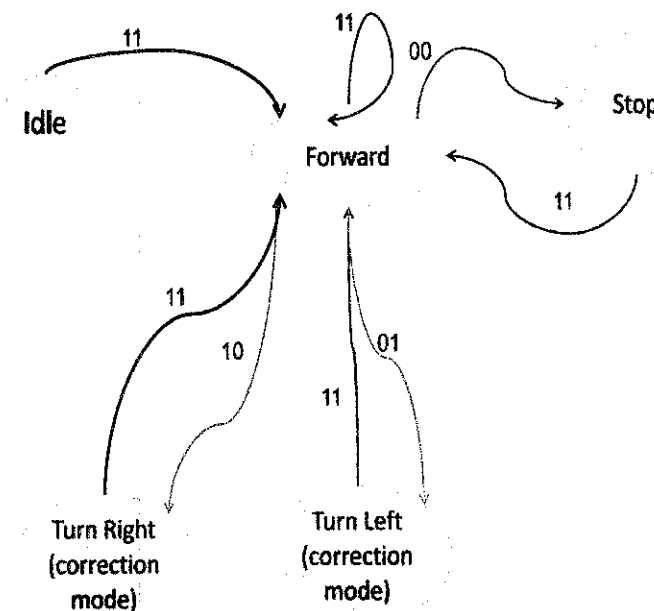


Figure 4: State transition diagram for simple line following behavior

All applications must have actions or outputs to be performed. The term '*output*' is commonly used for hardware while term '*action*' is for software. Several types of

actions such as entry action, exit action, input action and transition action. These actions can be defined depending on the conditions and moment they are performed. **Entry action** is done when the FSM enters a state. **Exit action** is done when the FSM leaves the state. **Transition action** is an action performed during the state change and is transition dependent. **Input action** is when an input or transition condition is true. Input actions are state independent and are used in any state.

Not all of these actions are practically used. Only some actions are used depending on the model used. The best known are Mealy and Moore models. Moore model generates only entry actions while Mealy model generates only input actions. Choosing the type of model suitable for usage depends on the application. Hardware systems are best using Moore model [7].

2.4. Pseudo-code

Pseudo-Code is simply a numbered list of instructions to perform some task. It is an artificial and informal language that helps programmers develop algorithms. Pseudo-code is a "text-based" detail algorithm design tool used before a programmer writes a computer program. Writing pseudo-codes includes the usage of while, do, for, if, switch. Examples below will illustrate this notion.

Example of pseudo-code:

If student's grade is greater than or equal to 60

Print "passed"

Else

Print "failed"

2.5. Conventional programming

Traditional sequential programs is structured as a single flow of control using standard constructs such as loops and nested function cells. This type of program represented in execution context in the location of the program counter and in the procedure call tree in the temporary variables allocated on the stack [7].

2.6. Event-driven programming

The grid follower robot's programming is an event-driven programming technique. Events are a better means of managing I/O concurrency in server software than threads: events help avoid bugs caused by the unnecessary CPU concurrency introduced by threads. Event-based programs also tend to have more stable performance under heavy load than threaded programs [8]. Event driven programming is a very flexible way of allowing programs to respond to many inputs or events. Unlike traditional programming, where the control flow is determined by the program structure, the control flow of event driven programs is largely driven by external events. Typically, event loops are pre-programmed to continually look for information to process.

Event-driven programs require detailed event-handler functions that must execute fast and always return to the main event-loop so no context can be preserved in the call tree and the program counter. Event-driven programming relies heavily on static variables to preserve the execution context from one event to the next [7].

One of the biggest challenges of using event-driven programming is on the management of the execution context. This execution context is represented as data and will be feed back into the control flow of the event-handler code. This allows each event handler to execute only the actions appropriate in the current context. However, this dependence on context data leads to deeply nested if-else constructs and this will become complicated to understand, hard to test and maintain. It will be easier to understand, test and maintain the program if a fraction of these conditional branches can be eliminated [7].

2.6.1. C and PBASIC Program

C provides the fundamental control-flow constructions required for well-structured programs. It is used to various applications. However, in this project, program used is the Parallax BASIC language (PBASIC) which is a language proprietary for Parallax mobile robots. Software used is the **Basic Stamp Editor v2.5.1**. This software translates the program into a PBASIC language that the Stamp understands, but would be very hard for humans to read and write then in this form the program is sent to the Stamp. This will leave room for the Stamp to do other things while the difficult process is done by the computer [9].

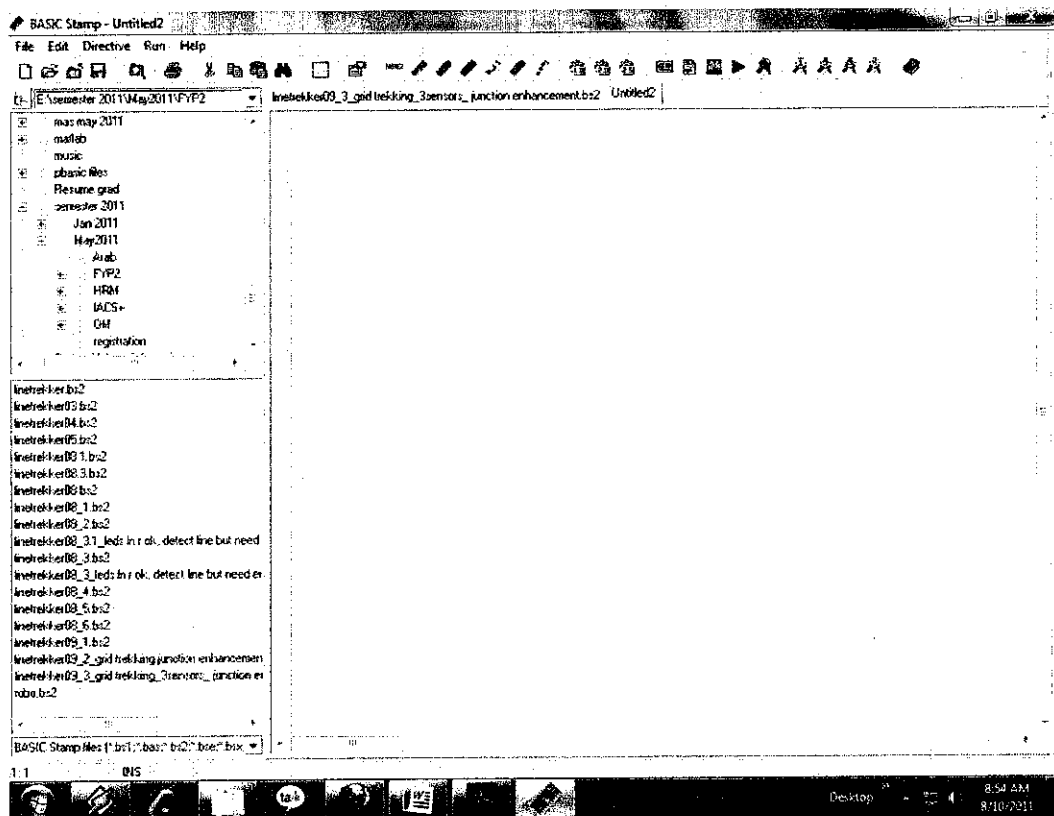


Figure 5: Basic stamp editor v2.5.1

CHAPTER 3

METHODOLOGY

3.1. Research Methodology

This research project is done by using the Parallax Inc's *Boe – BotTM* Basic Stamp 2 (BS2) module, Board of Education robot kit. The hardware is programmed using the BASIC Stamp Editor v2.5 which is using the PBASIC programming language. PIC16C57 is embedded on the BS2 module [4]. The grid follower robot is build based on the Parallax Boe-Bot Board of Education kit, line sensor consists of white LEDs and LDRs, Servo motors, and programming using PBASIC software. This project focuses on two main parts that are the grid following behavior modeling and the line sensor assembly.

3.1.1. Building the line sensor using Light Dependant Resistors (LDR) and white LEDs

Line sensor is a circuit which is able to see the black & white lines on the floor and translate the wheels location in relation to the line [5]. The sensor will detect the black tape on white surface. Light emitted by the white LED is reflected by the white surface while black surface absorbs the lights.

In order to let the robot sense the line, a sensor circuit has to be made. A low cost but functional sensor consists of LED-LDR combination sensors [4]. This is a circuit

which is able to see the black line on the floor as input and converts the vehicles location in relation to the line into a digital signal. If 2 LEDs are used, there might be a combination of 00, 01, 10 and 11 and these digital signals will determine the program that will be executed to keep the robot on the line. More sensors give more information to the mobile robot to locate itself on the grids.

Inputs		Description	
Left	Right	Sensor position on the track	Action
0	0	Both sensors detects black, robot at junction.	Stop
0	1	Left sensor on black, right sensor on white. Robot is not straight on the grids.	Turn left (Correcting position)
1	0	Right sensor on black, left sensor on white. Robot is not straight on the grids	Turn right (Correcting position)
1	1	Both sensors on white surface. Robot moves forward	Move forward

Table 2: Basic movements of the line following application

Robot navigation is the ability of the robot to perform movements. The robot will be programmed to perform movements such as moving forward, backward and turn. The first stage is programming the robot to perform the basic movements blindly without the sensors. Second stage is to program the robot to maneuver on lines with the input feed from the sensors.

3.1.2. Testing Sensor Response to light reflection from LED

```

LDRDetectLeft = IN3      'left led is assigned to I/O pin 3
LDRDetectRight = IN6     'left led is assigned to I/O pin 6

DEBUG CRSRXY, 0, 3, "L = ", BIN1 LDRDetectLeft,
                  "      R = ", BIN1 LDRDetectRight

```

Program 1: Testing sensor response

Above is the program codes used to display the input value detected at both LED-LDR sensors. The information will be displayed at the debug terminal. Debug terminal is small window displaying required information asked by user in the coding.

The line sensor operates based on the detection of light intensity. When light reflected from bright surface to the LDR the debug terminal will detect as '1' and vice versa. The combination of both bits (00, 01, 10 and 11) defines the state of movement the robot will perform. To test the response to light:

1. Enter, save, and run the line trekking program in Chapter 4.
2. Make sure it is still connected to the serial cable and that the measurements are displaying in the Debug Terminal.
3. Let the sensors head downwards. This is to ensure actual situation when light is reflected to it. Put a black paper or dark surface under the right LED-LDR couple. Observe the debug terminal and compare it with the movement of the servo.
4. Record the movement of the servo for every case is recorded.

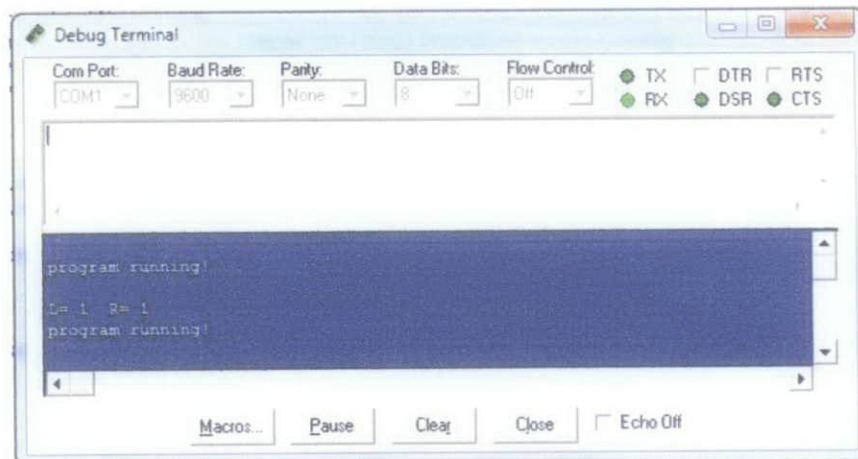


Figure 6: Both couple of LED-LDR sensor detects light

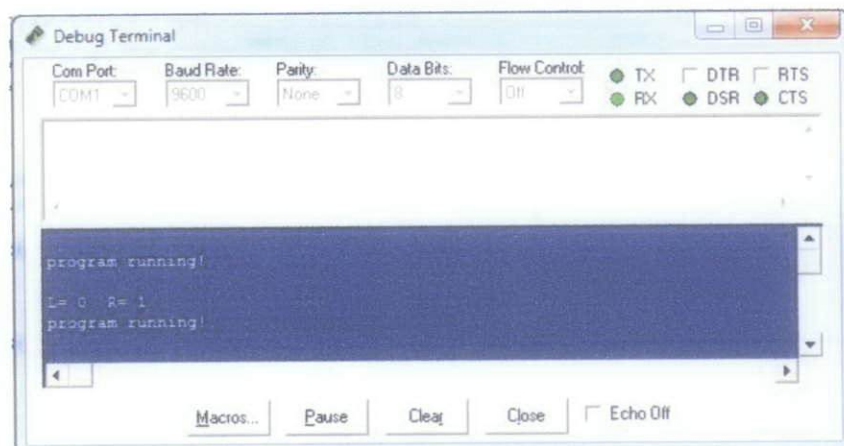


Figure 7: Left sensor does not detect light on a dark surface

Bits	State/ movement	Pulse duration		Servo rotation	
		S12	S13	Left servo	Right servo
00	Pauze (stop)	750	750	Stop	Stop
01	Turn left	850	850	CCW	CCW
10	Turn right	650	650	CW	CW
11	Forward	850	650	CCW	CW

Figure 8: Testing sensor response

3.1.2. Grid Following Behavior Modeling

The grid follower robot's programming is an event-driven programming technique. There are many benefits of using event-driven programming as explained in chapter 2. Event loops are pre-programmed to continually look for information to process. The robot will continually look for information from the sensor to be processed and converted to action.

The algorithm development will first be based on the Finite State Machine (FSM). FSM is a technique to design digital logic or computer programs using mathematical abstraction. It shows behavior model of a certain system and is written using finite number of states, the transition from the current state to another state and the actions taken by the states. It is quite similar to flow graph where the step by step sequence of the logic when certain conditions are met can be seen when it runs [2]. FSM is a brilliant method for human to understand and construct computer program.

The FSM will be developed into full algorithm which is in pseudo-code. The program will be further refined and written as PBASIC program. The PBASIC code is a switch-select type of coding. Thus, in the actual code, the main construct that will be used is switch-select because it closely neglects the format of FSM. Thus,

program development and debugging will be much easier compared to other methods.

There are basically two stages on the Grid Following behavior modeling which is firstly to model the line following behavior first, and then more states will be added to upgrade it to have the grid following behavior. This will be explained further in Chapter 4, Algorithm Development.

3.2. Key milestone

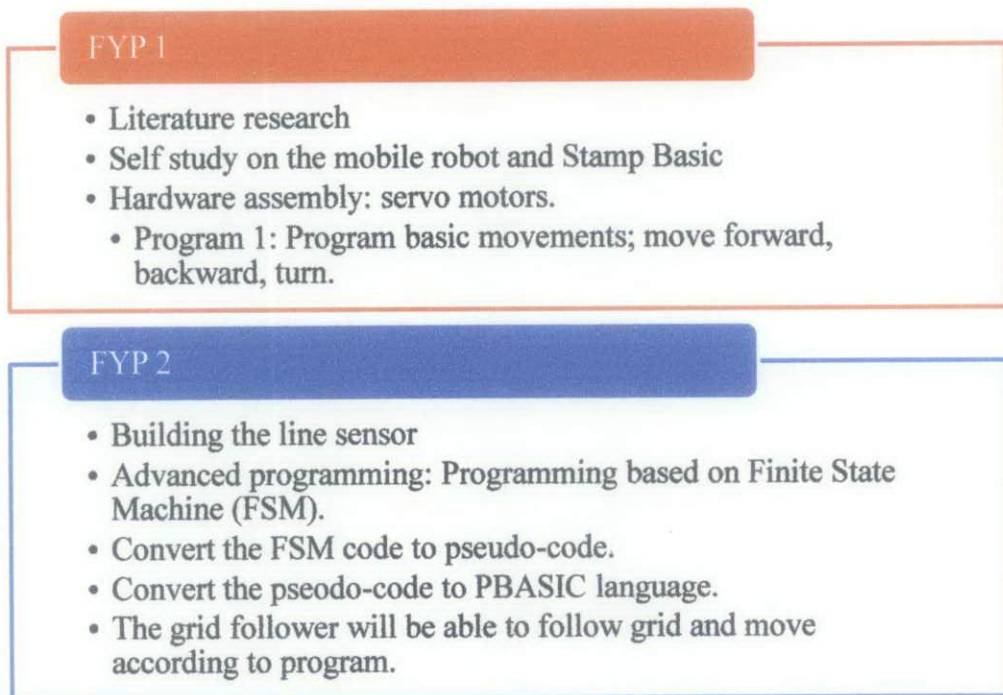


Figure 9: Key milestone of the project

3.3. Project activities

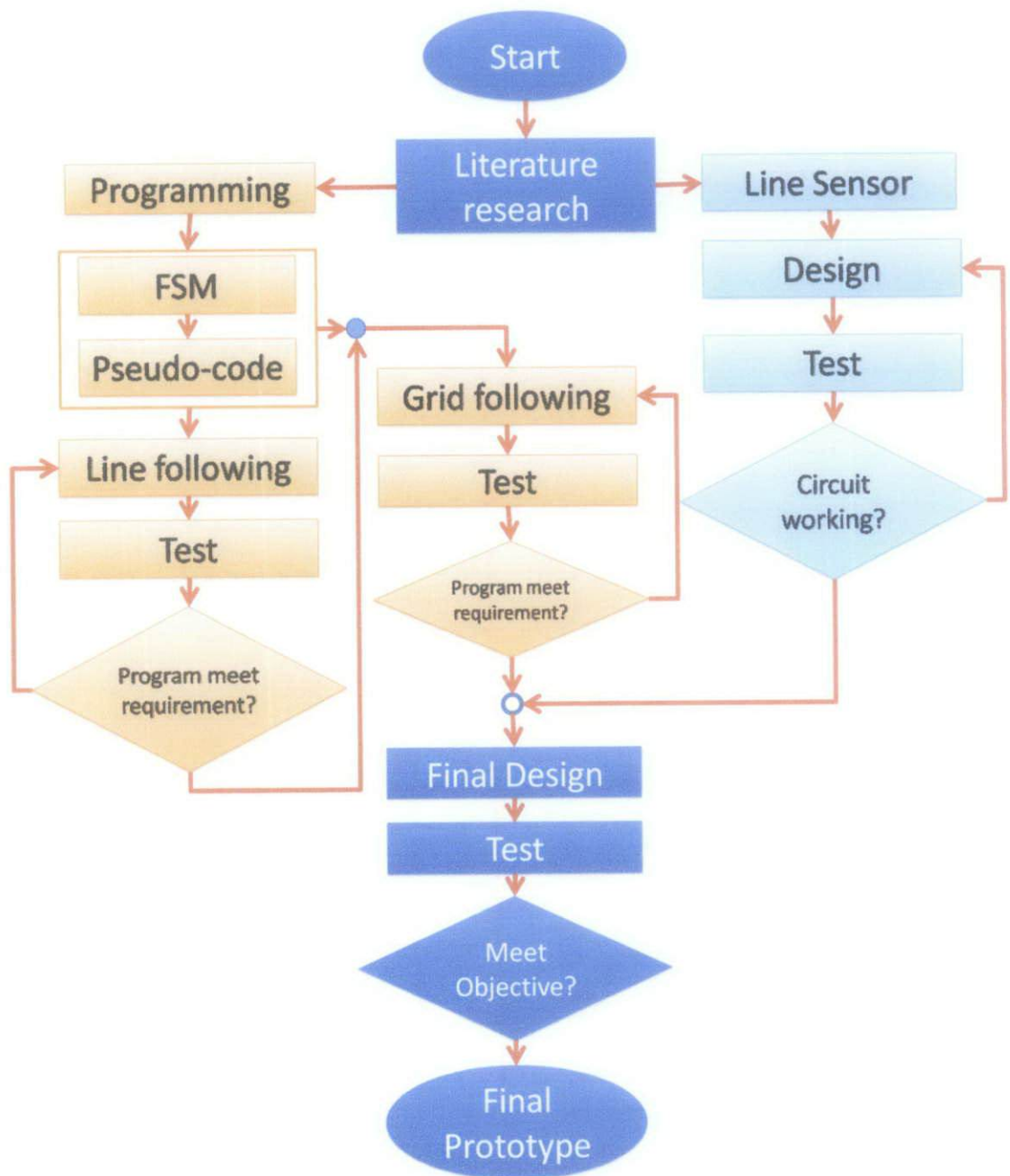


Figure 10: Activity flow

3.4. Tools used

3.4.1. Hardware

Boe-Bot Robot Parts Kit		
Parallax Stock Code	Description	Quantity
BS2-IC	BASIC Stamp 2 microcontroller module	1
28150	Board of Education Rev B	1
750-00008	300 mA 9 VDC power supply	1
800-00003	Serial cable	1
150-01020	1 k Ω resistors	2
150-02020	2 k Ω resistors	2
150-02210	220 Ω resistors	8
350-00006	Red LED	2
350-00009	Photoresistors (EG&G Vactec VT935G group B)	2
350-90000	LED standoff for infrared LED	2
350-00001	LED light shield for infrared LED	2
451-00303	3-Pin Header	2
700-00015	#4 screw-size nylon washer	2
710-00007	7/8" 4-40 pan-head screw, Phillips	2
713-00007	1/2" Spacer, aluminum, #4 round	2
800-00016	Jumper wires (bag of 10)	2
28133	Boe-Bot Hardware Pack :	1
700-00002	4-40 x 3/8" machine screw, Phillips	8
700-00003	Hex nut, 4-40 zinc plated	10
700-00009	Tail wheel ball	1
700-00016	4-40 x 3/8" flathead machine screw, Phillips	2

700-00022	Boe-Bot aluminum chassis	1
700-00023	1/16" x 1.5" long cotter pin	1
700-00025	13/32" rubber grommet	2
700-00028	4-40 x 1/4" machine screw, Phillips	8
700-00038	Battery holder with cable and barrel plug	1
700-00060	Standoff, threaded aluminum, round 4-40	4
721-00001	Parallax plastic wheel	2
721-00002	Rubber band tire	4
900-00008	Parallax Continuous Rotation Servo	2

Table 3: Hardware used based on the parallax Boe-Bot student guide [ERROR! BOOKMARK NOT DEFINED.]

3.4.2. Software

- Parallax BASIC Stamp Software v2.5

3.4.3. Computer system requirement:

- Windows 98 or newer operating system
- A serial or USB port
- A CD-ROM drive, World Wide Web access, or both

CHAPTER 4

RESULT AND DISCUSSION

4.1. Hardware assembly

For this project, the Grid Trekker robot is assembled using Parallax Inc. Board of Education. This kit comes with a microcontroller module named Basic Stamp 2 (BS2) which has a microcontroller PIC16C57 embedded on the module.

- Voltage source: 6V DC
- Program is transferred to microcontroller via serial port.

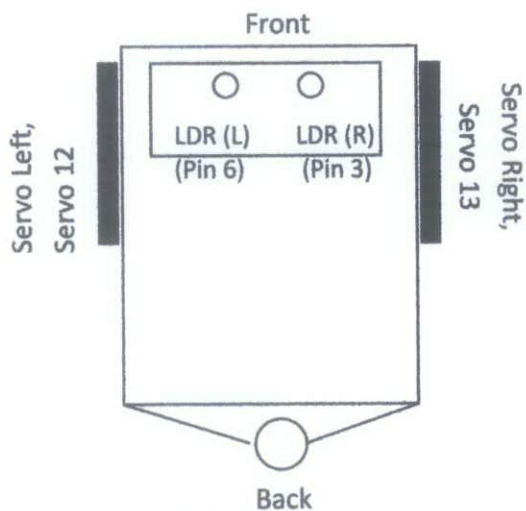


Figure 12: Pins and servo assignment

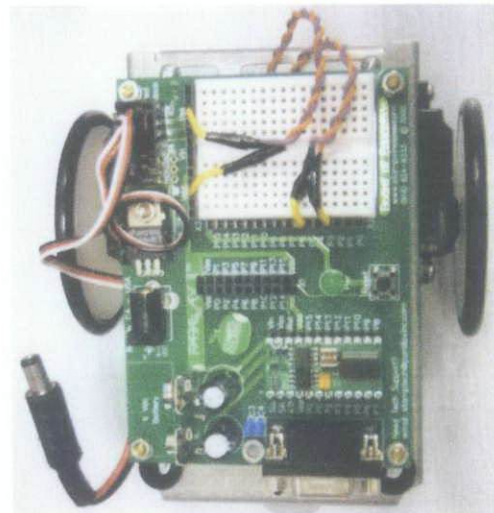


Figure 11: Parallax Boe-Bot



Figure 13: 6v power source

4.1.1. Servo Motors

One of the important parts of the mobile robot is the servo motors. Mobile robots need to use continuous servo motors or also known as 360° DC motors. The mistake done during executing this project is using the wrong servos. Using 180° servos does not give the correct response of the robot's movement. This is because Grid Trekker robot must have the capability to turn various angles. Thus, using 180° servos limits the angle to 180° turns only.

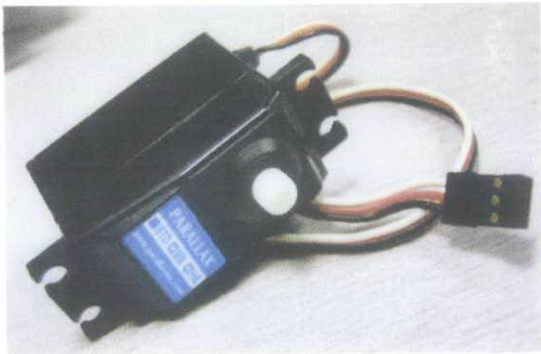


Figure 15: servo motor

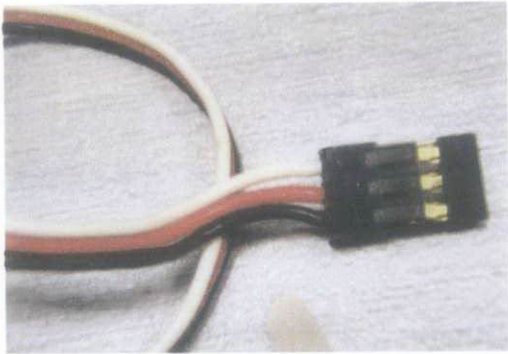


Figure 14: 3 pins plug servo motor

4.1.2. Line sensor construction

Circuit construction of the line sensor is as the diagram. A simple LED-LDR combination sensor is constructed. A circuit with two pairs of LED-LDR sensors is built. Using two pairs of LED-LDR sensors will reduce the power consumption.

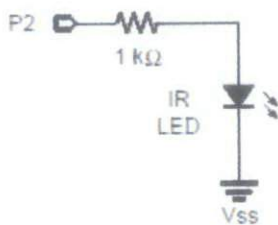


Figure 16: LED sensor circuit diagram [4]

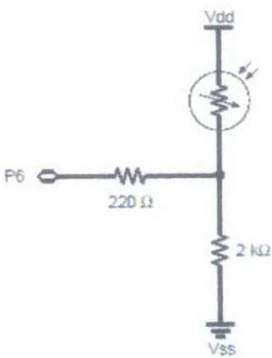


Figure 17: LDR sensor circuit diagram [4]

Components Used For Line Sensor Circuit:

No.	Components	Quantity
1	2k Ω resistor	2
2	1k Ω resistor	2
3	220 Ω resistor	2
4	White LED	2
5	Heat shrink tubing	5cm
6	Single core wire	As needed.
7.	Veroboard	1

Table 4: Components for line sensor

Equipments used:

1. Solder
2. Solder iron
3. Flux
4. Wire cutter

Circuit is constructed and tested on breadboard before permanently mounted on the veroboard. One of the factors that needed to be taken care of when building a circuit is the distance between both pairs of LED-LDR sensors. The distance should not be too wide or too close and this depends on the width of the grids track. The distance used for this line sensor is 2.5cm.

Once the circuit is setup on the veroboard, testing is done using the Parallax Basic Stamp Editor V2.5.1. See the coding in *Chapter 3 (3.1.2 Testing Sensor Response to Light Reflection from LED)*.

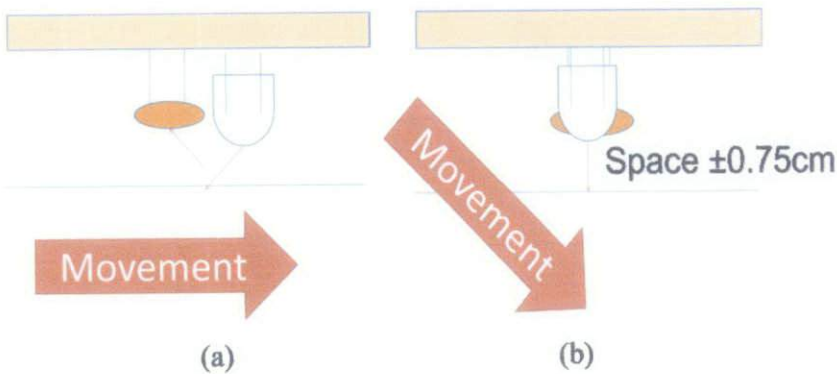


Figure 18: (a) Side view of the line sensor, (b) Front view of the line sensor

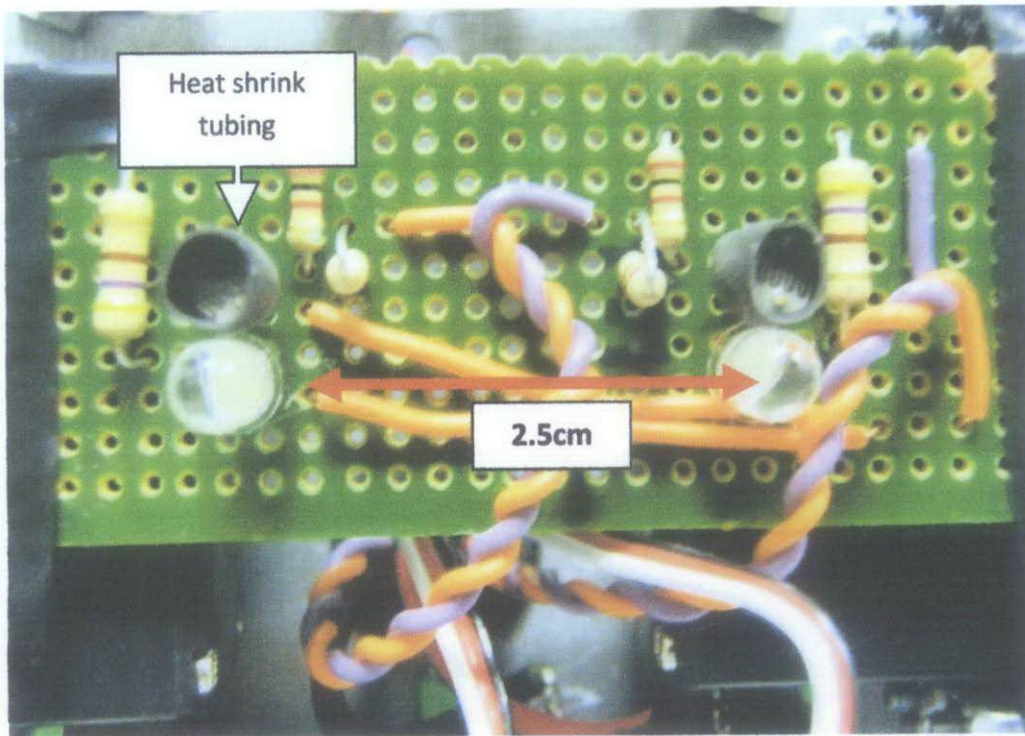


Figure 19: Line sensor circuit consisting of LED-LDR

Circuit is then soldered on the veroboard. This process needs to be carried out carefully since the components easily burnt if heat from the solder is applied directly to the components. When soldering, heat should be applied to the solder iron and then apply the melted iron to the components leg. During the soldering process, heat from the solder must not have direct contact to the components legs for too long as it may burn the components. Current across the components are measured to ensure the circuit functionality. Zero current indicates shorted circuit and vice versa. Solder iron is applied only to the necessary point cleanly and not connected to other points on the veroboard as this may cause shorted circuit as well. Connection is checked and ensure the veroboard is slashed at certain points (as shown in figure 20) to separate both sensor circuits from one another to prevent short circuit.

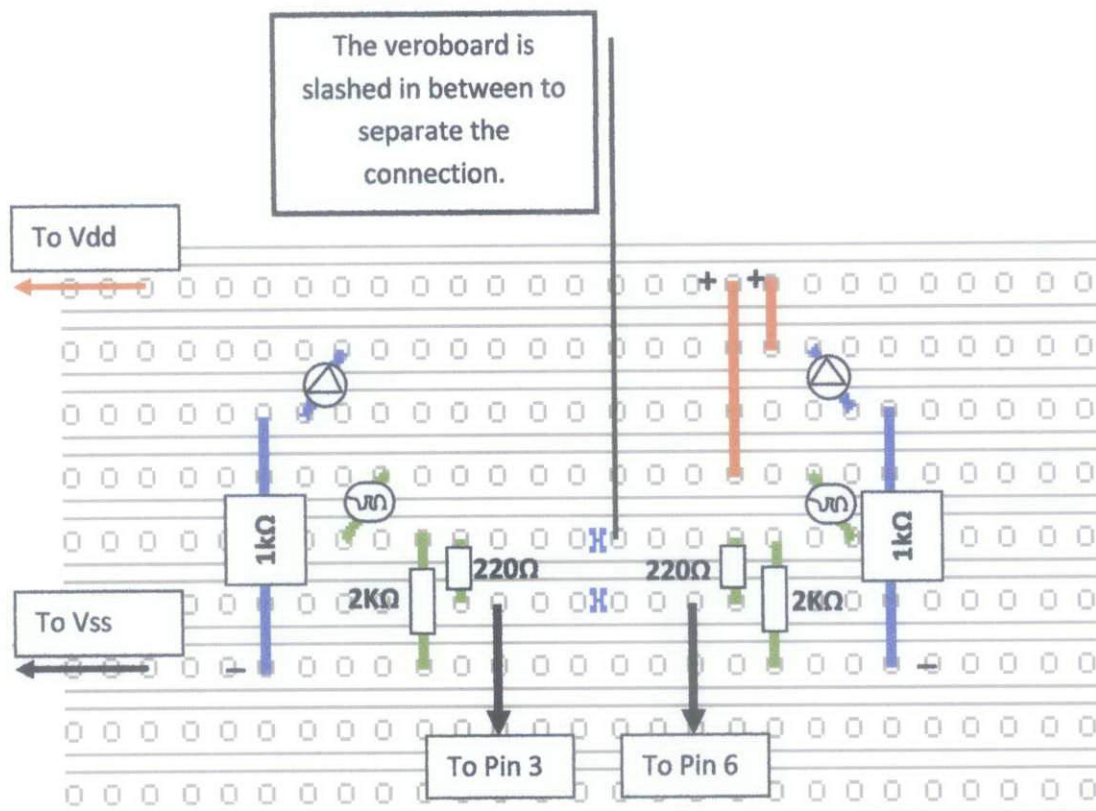


Figure 20: Veroboard line sensor circuit

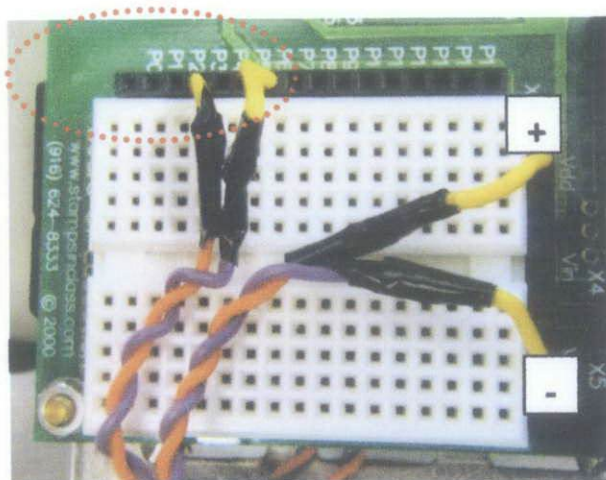


Figure 21: Sensor assigned to I/O pin3 and pin6, power source connected

The output from the circuit will be connected to I/O pin 3 and I/O pin 6 of the basic stamp. I/O stands for input/output. The BASIC Stamp has 24 pins and 16 of them are I/O pins. This I/O pins acts as the intermediate between the microprocessor and external devices; for example the line sensor. Sensor will give input from the

environment to the microprocessor to be monitored. This enables the I/O pins programmed to detect and react to the external input. Only with external inputs enable Grid Trekker robot to indicate the grids on the track based on light detection. This gives the robot a sense of intelligence.

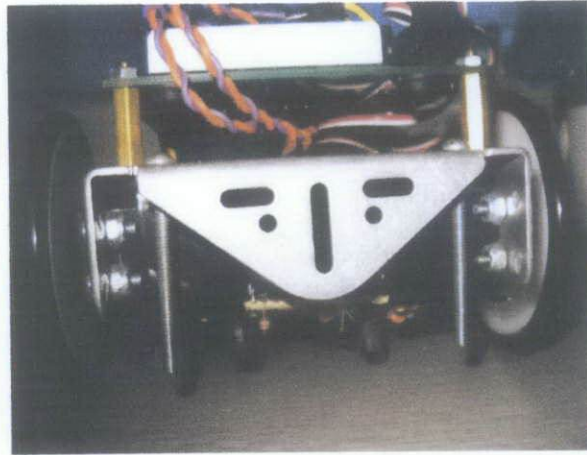


Figure 22: Sensor is mounted under the grid follower robot

4.1.2.1. Sensor drawbacks

The LED-LDR line sensor is low cost and simple. As low-cost as it is, one cannot get an accurate input sensing from it. One of the problems caused from this cheap line sensor is the sensitivity of the LDR cannot be controlled. LDR is sensitive to the environmental light that disturbs the input sensing. Covering the LDRs with heat shrink tubing is one of the solutions. Heat shrink tubing is a material used to protect wirings in which this tubing shrinks when heat applied. Thus, LDR can be covered from always detecting surrounding lights especially the light from the LED. Another weakness of this line sensor is it provides slow response. Speed of the Grid Trekker robot must be put at the slowest due to the underperformance of the line sensors.

4.1.2.1.1. Sensor Position

Sensor position also affects the accuracy of the Grid Trekker's movement. Sensor must be put in the middle of the servos. Putting it at the front or back will cause

failure of turning accurately at the junction. Due to slow response of sensor, the Grid Trekker robot will stop at the junction with some displacement. Putting sensor at the middle counters the problem.

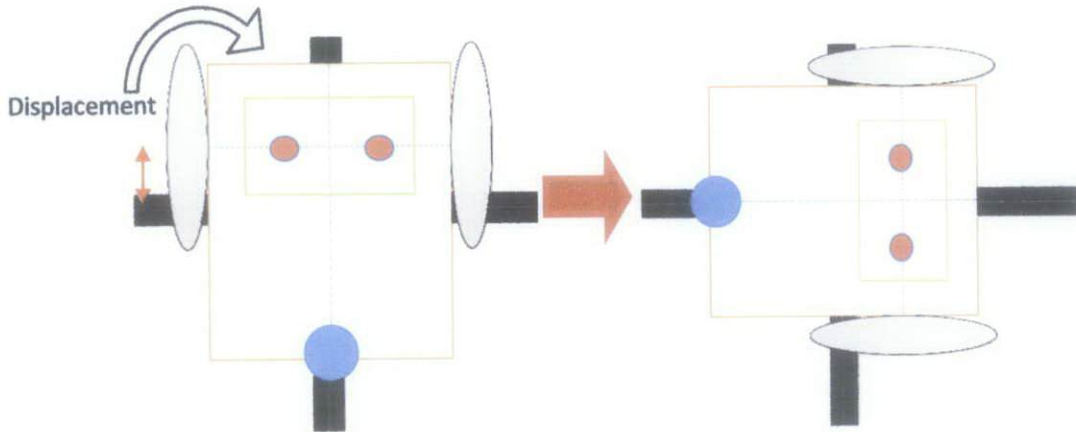


Figure 23: Position of sensor affects accuracy of turn at junction

4.1.2.1.2. LEDs as Indicators to Verify Input Detection

Problem occurs where the Grid Trekker robot cannot detect junctions properly. To observe the action, programmer need to know what exactly is the input detected by the sensors. It is difficult to observe the action without LED indicator. Thus, a simple LED circuit is constructed on the mini breadboard. The indicators include

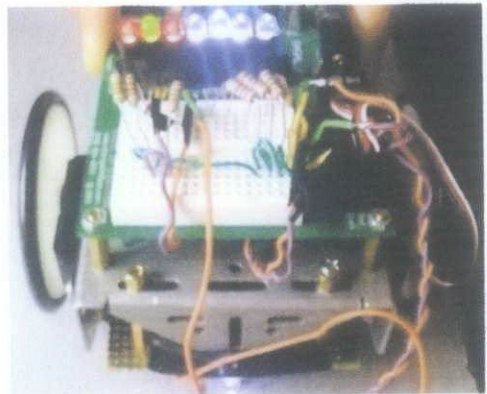


Figure 24: LED indicators added

detecting left and right, and indicator for number of junctions. LEDs are assigned to other free I/O pins and a subroutine for this function is added to the program. See the program in *Chapter 3 (3.1.2 Testing Sensor Response to Light Reflection from LED)*.

4.1.2.1.3. Number of Sensors

The next problem faced is when the Grid Trekker robot fails to have a precise turn at the junction. Number of sensors used affects accuracy of the robot

positioning. As only a pair of sensors are used, the Grid Trekker robot maintain its position on exactly to the middle of the line. While the Grid Trekker robot is not at the middle of the line and detects a junction, it will not have an accurate turn. By adding another sensor this problem can be countered. There are two options of adding sensors either in between the two current sensors, or at the front or back of the Grid Trekker robot.

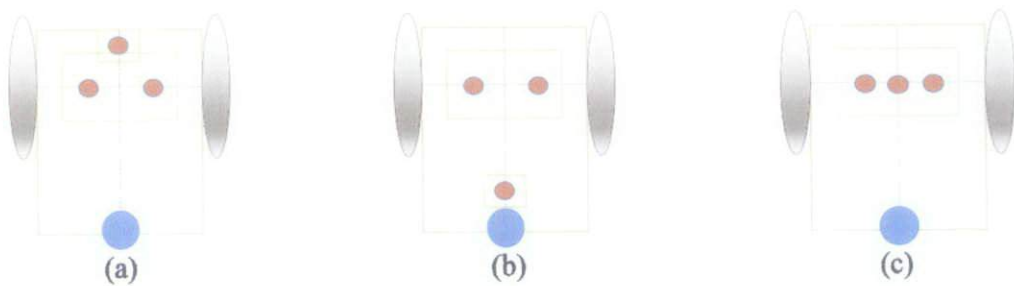


Figure 25: Possible position for the third sensor; (a) At the front (b) At the back, (c) Between the existing two

4.2. Building the Grid Track

The track is built using PVC black tape on a white surface. Note that surface and the tape itself must not reflect light as this will produce disturbance to the sensor in acquiring data based on the reflection of on the grid track. Black tape will absorb most of the light beamed by the LEDs while bright surface (white) reflects most of the light and sensed by the LDR.

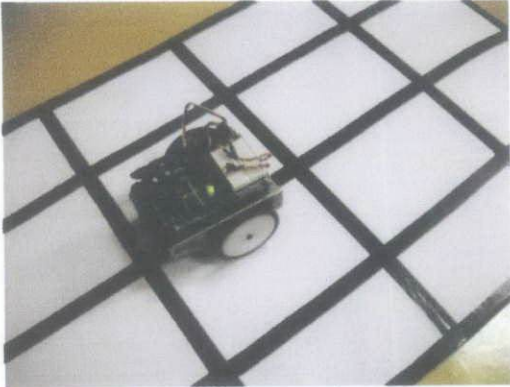


Figure 26: Grid follower robot maneuver according to grids

4.3. Algorithm Development

Finite State Machine gives clearer overview on how to program the movements of the Grid Trekker robot. FSM is a powerful method for behavior modeling and has been explained in chapter 2. The process of software modeling is first software

modeling is done using FSM model. Then pseudo code will be written based on the FSM before it is converted to PBASIC program. There are two stages in developing the algorithm for Grid Trekking which is first from the development of line following algorithm then only adding some states for grid trekking.

4.3.1. Line Following

4.3.1.1. Line following Behavior

Figure 27 shows basic line following behavior.

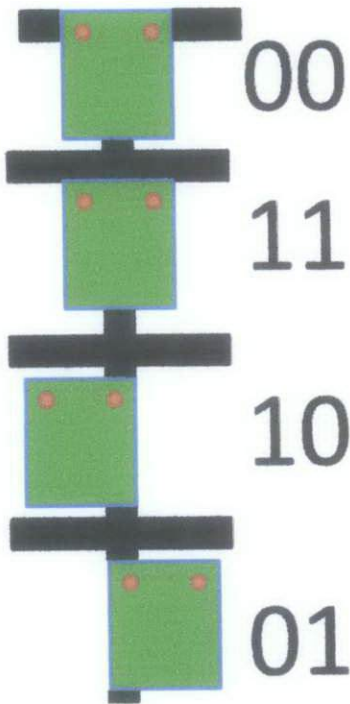


Figure 27: Line following robot behavior

Line following behavior overview

When both sensors are on black line (00), consider it at a junction of the grid. Mobile robot stops at junction.

When both sensors are on white surface (11), consider it in a position straight on the black line and it will move forward.

When the mobile robot position is not straight on the black line, either one of the sensors will detect the black line.

When the left on white (1) and the right sensor on black line (0), the robot will have to correct its position straight on the black line by turning right until its position is straight and vice versa for the other case (10).

Table 5: Line following behavior overview

4.3.1.2. State Transition Matrix for Line Trekker

Recall from Chapter 2, state transition matrix is a tool for FSM model. To achieve grid trekking behavior, line trekking behavior modeling is where the work starts with. Based on the table 6 below, there are only five states used for line trekking. More explanation can be found in *Chapter 2, State Transition Matrix*.

Present states\Next states	00	01	10	11
Idle	-	-	-	Forward
Forward (11)	Stop	Turn left	Turn right	Forward
Turn Left (01)	Stop	Turn left	Turn right	Forward
Turn Right (10)	Stop	Turn left	Turn right	Forward
Stop (00)	Stop	Turn left	Turn right	Forward

Table 6: Transition matrix for simple line trekker behavior (from chapter 2)

4.3.1.3. Drawing State Chart Pseudo code

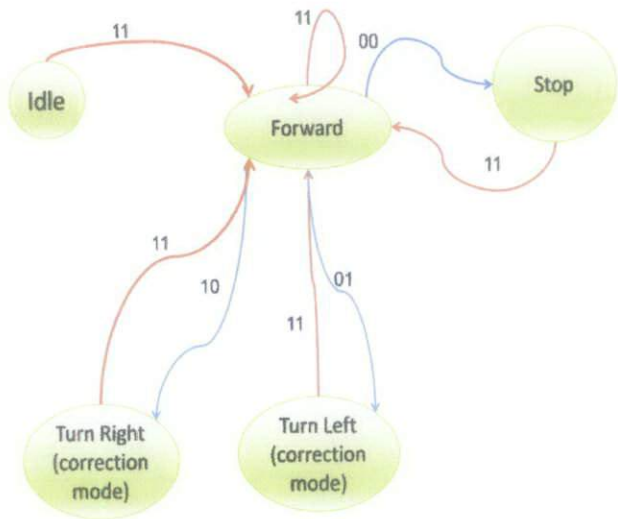


Figure 28: State transition diagram for simple line following behavior from chapter 2

4.3.1.4. Pseudo code for Line Trekking

Based on (A) and (B) programs below, event-driven programming is more comprehensible. Even though the conventional program is shorter, the program may be confusing to be understood especially when the number of states increases. Thus, conventional programming is not suitable for complex behavior modeling. Using event-driven technique instead, manages the program to its behavior or states making it more comprehensible. Event-driven technique enables programming for more complex and realistic behavior.

A. Conventional

```

DO
Assign left and right LDR Sensor to I/O Pins
Open debug terminal for left and right LDR sensors

  if Left LDR detected then
    if Right LDR detected then          ' L = 1, R = 1
      go to subroutine forward
    else                                ' L = 1, R = 0
      go to subroutine turn right
    endif
  else
    if Right LDR detected then          ' L = 0, R = 1
      go to subroutine turn left
    else
      go to subroutine stop
    endif
  endif
LOOP

```

Pseudo code 1: Line Following Program Using Conventional Programming (main routine).

B. Event-Driven

```

DO
Assign left and right LDR Sensor to I/O Pins
Open debug terminal for left and right LDR sensors

if both Left and Right LDR not detected then
state=forward
endif

CASE forward
  go to subroutine forward
  if both Left and Right LDR not detected then
    state=pauze

ELSEIF Left LDR not detected and Right LDR detected then
state=tleft

```

```

SELECT state

CASE idle
    IF Left LDR not detected AND Right LDR not detected THEN
        state=forward
    ENDIF

CASE forward
    go to subroutine goforward
    IF Left LDR not detected AND Right LDR not detected THEN
        state is pauze

    ELSEIF Left LDR not detected AND Right LDR detected THEN
        state is tleft

    ELSEIF Left LDR detected AND Right LDR not detected THEN
        state is tright

    ENDIF

CASE pauze
    go to subroutine GoStop
    IF Left LDR not detected AND Right LDR Detected THEN
        state is tleft
    ENDIF

CASE tleft
    go to subroutine TurnLeft
    IF Left LDR detected AND Right LDR Detected THEN
        state is forward
    ENDIF

CASE tright
    go to subroutine TurnRight
    IF Left LDR detected AND Right LDR Detected THEN
        state is forward
    ENDIF

ENDSELECT
LOOP

```

Pseudo Code 2: Event Driven Pseudo Code.

4.3.1.5. Line Following Program

```
DO
LDRDetectright = IN3           'assign input to pin 3 and pin 6
LDRDetectleft = IN6

DEBUG CRSRXY, 0, 3, "L= ", BIN1 LDRDetectLeft,
" R= ", BIN1 LDRDetectRight

IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
state=forward
ENDIF

CASE forward
GOSUB GoForward
IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
state=pauze

ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
state=tleft

ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
state=tright

ENDIF

CASE pauze
GOSUB GoStop
IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
state=tleft
ENDIF

CASE tleft
GOSUB TurnLeft
IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
state=forward
ENDIF

CASE tright
GOSUB TurnRight
IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
state=forward
ENDIF

ENDSELECT
PAUSE 20
LOOP
```

Program 1: Line Following Grid Trekking Main Routine PBASIC Program.

4.3.1.6. The Subroutines for Line Trekking

```
'-----[subroutine_idlez]-----
idlez:
IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
PULSOUT 13, 750
PULSOUT 12, 750
PAUSE 20
ENDIF
RETURN
'-----[subroutine_GoStop]-----
GoStop:
FOR counter = 1 TO 8
PAUSE 20
PULSOUT 13, 750
PULSOUT 12, 750
PAUSE 20
NEXT
RETURN
'-----[subroutine_GoForward]-----
GoForward:
PAUSE 20
PULSOUT 13, 650
PULSOUT 12, 850
PAUSE 20
RETURN
'-----[subroutine_TurnRight]-----
TurnRight:
PAUSE 20
FOR counter = 0 TO 1
PULSOUT 13, 850      's12 is left
PULSOUT 12, 850
PAUSE 20
GOSUB GoStop
NEXT
RETURN
'-----[subroutine_TurnLeft]-----
TurnLeft:
PAUSE 20
FOR counter = 0 TO 1
PULSOUT 13, 650      's13 is right
PULSOUT 12, 650
PAUSE 20
GOSUB GoStop
NEXT
RETURN
```

Program 2: Line Trekking Subroutines

To ensure program is neat and manageable, subroutines are used. Subroutine is a portion of code within a larger program that performs a specific task and is relatively independent of the remaining code. Subroutines can be called several times from several places during execution of a single program and then return back to the next instruction after the subroutine is done. This will reduce redundancy of the program as well. For line trekking, there are five subroutines, each represent each state present for the program.

4.3.2. Grid Following

The difference of grid following to line following is grid following behavior must be able to turn 90° left or right (depending on the programmer's formation). A grid following robot must be able to perform movements according to the formation set. This means it must be able to count junctions and can distinguish which junction to turn 90° .

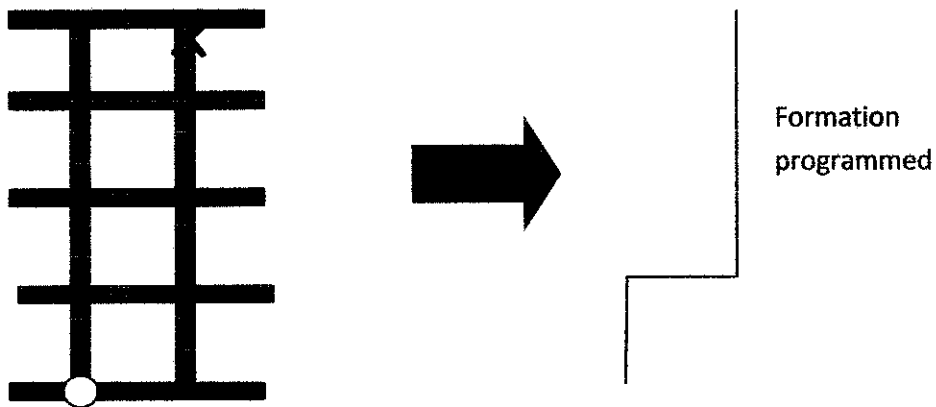


Figure 29: Mobile robot able to maneuver on grids according to the formation set

4.3.2.1. Grid following Behavior

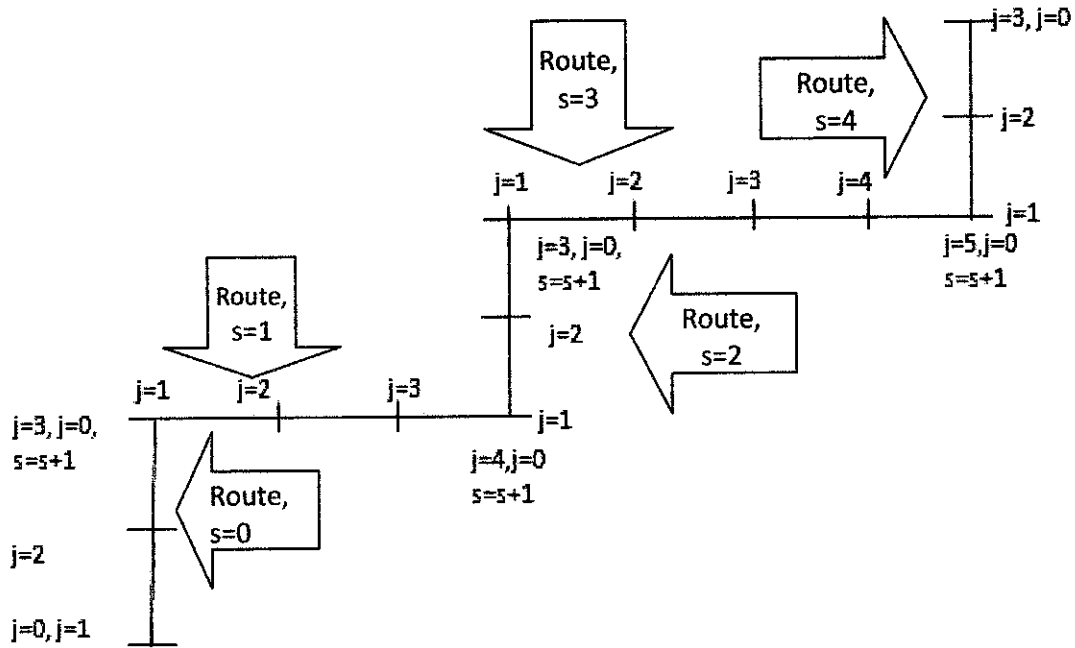


Figure 30: Grid trekking behavior

Figure 30 shows an example of the formation that may be programmed to the Grid Trekker. Table 7 shows the behavior of the Grid Trekker in which the movements are based on what is programmed. The difference of Grid Trekking to Line trekking is Grid Trekking requires the usage of arrays and counters in order to count and give the robot memory and intelligence on where should the Grid Trekker turn at junction. Other addition is the action of turning 90 degrees when it reaches a junction where Line Trekking does not have.

Route (s)	Junction (j)	Behavior
0	J=1 J=2, 0	At route s=0, junction is counted until reaches j=2 then reset. S is then incremented to a new route. When this condition reached, the grid trekker will turn right 90 degrees.
1	J=1, 2 J=3, 0	At route s=1, junction is counted until reaches j=3 then reset. S is then incremented to a new route. When this condition reached, the grid trekker will turn left 90 degrees.
2	J=1 J=2, 0	At route s=2, junction is counted until reaches j=2 then reset. S is then incremented to a new route. When this condition reached, the grid trekker will Stop.
3	J=1, 2,3 J=4, 0	At route s=3, junction is counted until reaches j=3 then reset. S is then incremented to a new route. When this condition reached, the grid trekker will Stop.
4	J=1 J=2, 0	At route s=4, junction is counted until reaches j=2 then reset. S is then incremented to a new route. When this condition reached, the grid trekker will Stop.

Table 7: Grid trekking behavior

Length of the route can be programmed based on number of junctions, j encountered. Combination of routes, s enables the grid trekker robot moves in a formation assigned. This detail is defined in the program. For further understanding, follow the state transition matrix.

4.3.2.2. State Transition Matrix for Grid Trekker

Current State	Entry Actions 1				Next State 1/ Direction (D)	Entry Actions 2				Next State 2
	L	R	j	s		L	R	j	s	
Idle	1	1	$j = 0$	$s = 0$	Forward	1	1	$j = 0$	$s = 0$	Forward
Forward	1	1	$j = 0$	$s = 0$	Forward	1	1	$j = 0$	$s = 0$	Forward
Forward	1	0	$j = 0$	$s = 0$	Turn Right	1	1	$j = 0$	$s = 0$	Forward
Forward	0	1	$j = 0$	$s = 0$	Turn Left	1	1	$j = 0$	$s = 0$	Forward
Forward	0	0	$j < L_1$	$s = 0$	Forward	1	1	$j = j + 1$	$s = 0$	Forward
Forward	0	0	$j = L_1$	$s = 0$	Turn Right 90°	1	1	$j = 0$	$s = s + 1$	Forward
Forward	0	0	$j < L_2$	$s = 1$	Forward	1	1	$j = j + 1$	$s = 1$	Forward
Forward	0	0	$j = L_2$	$s = 1$	Turn Left 90°	1	1	$j = 0$	$s = s + 1$	Forward
Forward	0	0	$j < L_3$	$s = 2$	Forward	1	1	$j = j + 1$	$s = 2$	Forward
Forward	0	0	$j = L_3$	$s = 2$	Turn Right 90°	1	1	$j = 0$	$s = s + 1$	Forward
Forward	0	0	$j < L_4$	$s = 3$	Forward	1	1	$j = j + 1$	$s = 3$	Forward
Forward	0	0	$j = L_4$	$s = 3$	Turn Left 90°	1	1	$j = 0$	$s = s + 1$	Forward
Forward	0	0	$j < L_5$	$s = 4$	Forward	1	1	$j = j + 1$	$s = 4$	Forward
Forward	0	0	$j = L_5$	$s = 4$	Turn Right 90°	1	1	$j = 0$	$s = s + 1$	Forward
Forward	0	0	$j = 0$	$s = s_{max}$	Stop	-	-	-	-	Stop

Table 8: State transition matrix for grid trekking

S	Length (S)		Direction (S)= D
0	L_1	$j=2$	Turn 90° Right
1	L_2	$j=3$	Turn 90° Left
2	L_3	$j=2$	Turn 90° Right
3	L_4	$j=4$	Turn 90° Left
4	L_5	$j=2$	Turn 90° Right

Table 9: Arrays

Recalling from Chapter 2, **entry action** is done when the FSM enters a state. Machines usually use Moore model where this model generates only entry actions. The FSM will only change from one state to another if the entry action of the current state is fulfilled. Referring Table 8, when the current state is 'Idle', the FSM will only go to the next state that is 'Forward' when left LDR (L) detects 1 and right LDR (R) detects 1 while junction detected (j) is by default zero and route (s) is zero. Else, the FSM will stay in the current state it is in. This applies with other states also.

4.3.2.3. Drawing State Chart

Similar to *Chapter 2*, the state chart of line following robot behavior has been explained. In this part, the focus is the next stage of program development; that is the grid following robot behavior modeling. Note that the difference of line following and grid following is the latter have more states. There are some modifications added for grid following behavior modeling that is 'Turn 90° Left' and 'Turn 90° Right' when it finds the junction. This involves counters for junction and route counting.

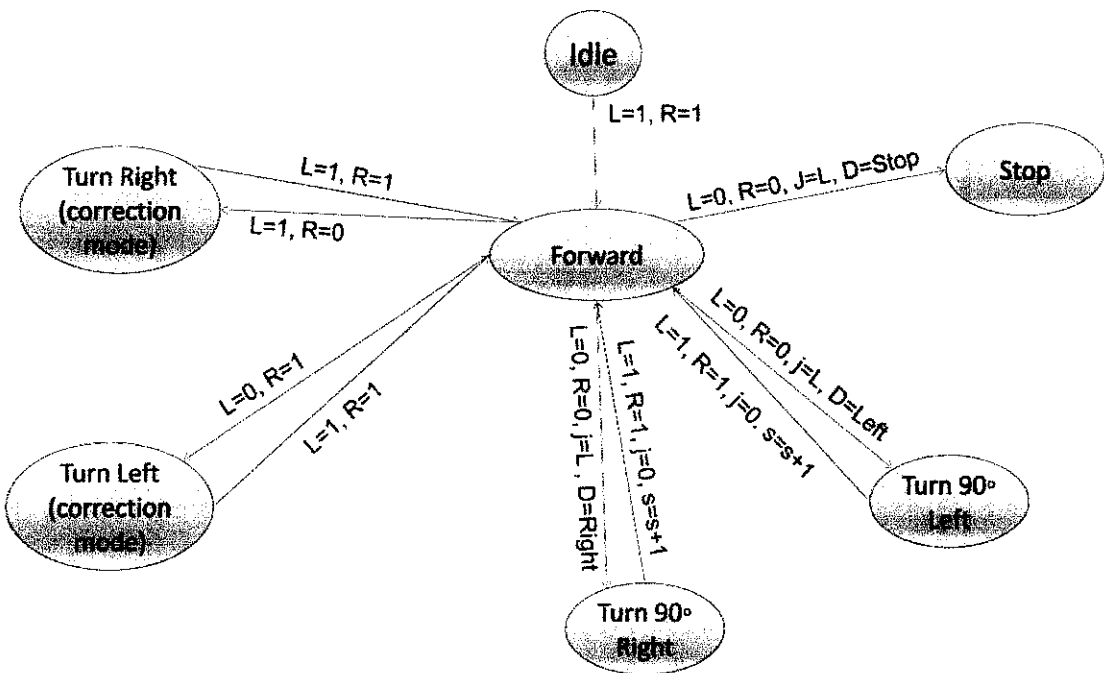


Figure 31: FSM state chart for grid follower robot

4.3.2.4. Pseudo code

```
'-----[main routine]-----
DEBUG: program running
  On LEDright at sensor
  On LEDleft at sensor
'-----[define route]-----
'set array for route length=no of junction encountered, j:
' length(s) = no. of junctions
length(0) = 2 junctions encountered
length(1) = 3 junctions encountered
length(2) = 2 junctions encountered
length(3) = 4 junctions encountered
length(4) = 2 junctions encountered
'set array direction(s)= response/direction of movement
's=route
direction(0) = turn right 90 degrees at junction
direction(1) = turn left 90 degrees at junction
direction(2) = turn right 90 degrees at junction
direction(3) = turn left 90 degrees at junction
direction(4) = in stop position

DO
  assign right LDR sensor to I/O pin 3
  assign left LDR sensor to I/O pin 6
  DEBUG:print both sensor L and R inputs to debug terminal

SELECT state

CASE 1: idle
  IF Left LDR detected AND Right LDR detected THEN
    state = forward
  ENDIF

CASE 2: forward
  Go to subroutine move forward
  IF Left LDR not detected AND Right LDR not detected THEN
    Increment junction encountered by one
    IF junction encountered = array_length(s) THEN
      array_direction(s)
      Increment s by one to move to the next route
    ENDIF
  ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
    Turn left to correct position
  ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
    Turn right to correct position
  ENDIF
```

```

CASE 3: turn left to correct position
    go to subroutine turnleft to correct position
    IF Left LDR detected AND Right LDR detected THEN
        state=forward
    ENDIF
CASE 4: turn right to correct position
    go to subroutine, turn right to correct position
    IF Left LDR detected AND Right LDR detected THEN
        state=forward
    ENDIF
CASE 5: turn left 90 degrees at junction
    go to subroutine, turn left 90 degrees at junction
    IF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
        state=turn right to correct position
    ELSEIF Left LDR not detected AND Right LDR detected THEN
        state=turn left to correct position
    IF Left LDR detected AND Right LDR detected THEN
        state=forward
    ENDIF
CASE 6: turn right 90 degrees at junction
    go to subroutine turn right 90 degrees at junction
    IF Left LDR detected AND Right LDR not detected THEN
        state=turn right to correct position
    ELSEIF Left LDR not detected AND Right LDR detected THEN
        turn left to correct position
    ELSEIF Left LDR detected AND Right LDR detected THEN
        state=forward
    ENDIF
CASE 7: stop position
    EXIT
ENDSELECT
LOOP
END

'-----[subroutines]-----
subroutine move forward
subroutine turnleft to correct position
subroutine, turn right to correct position
subroutine, turn left 90 degrees at junction
subroutine turn right 90 degrees at junction
subroutine stop position

```

Pseudo Code 3: Grid Trekking Main Routine Pseudo Code. Refer full program at APPENDIX C-b(ii)

4.3.2.5. Grid Following Program

```
'-----[main routine]-----
DEBUG CR,"program running!"
  HIGH LEDright
  HIGH LEDleft
'-----[define route]-----
length(0) = 2
length(1) = 3
length(2) = 2
length(3) = 4
length(4) = 2
direction(0) = trightj
direction(1) = tleftj
direction(2) = trightj
direction(3) = tleftj
direction(4) = pauze
directionB(0) = trightjf
directionB(1) = tleftjf
directionB(2) = trightjf
directionB(3) = tleftjf

seg = 4           'number of total segments in this route
j = 0             'reset junction counter
s = 0             'start at first segment
state = 0         'added 'to initialize state to zero

'define inputs
INPUT 3
INPUT 6
INPUT 5

DO
  LDRDetectright = IN3
  LDRDetectleft = IN6

  DEBUG CRSRXY, 0, 3, "L= ", BIN1 LDRDetectLeft,
    " R= ", BIN1 LDRDetectRight,

GOSUB SensorIndicator

SELECT state

CASE idle
  IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF
```

```

CASE forward
  GOSUB GoForward
  IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
    j = j + 1          'increment junction counter
  IF (j = length(s)) THEN
    HIGH LEDFront
    state = direction(s)
    j = 0
    s = s+1            'next segment
  ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
    state=tleft
  ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
    state=tright
  ENDIF
ENDIF

CASE tleft
  GOSUB TurnLeft
  IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

CASE tright
  GOSUB TurnRight
  IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

CASE tleftj
  GOSUB TurnLeftJ
  IF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
    state=tright
  ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
    state=tleft
  ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

CASE trightj
  GOSUB TurnRightJ
  IF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
    state=tright
  ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
    state=tleft
  ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

```

```
CASE pause
    EXIT
```

```
ENDSELECT
```

```
    PAUSE 20
LOOP
```

```
END
```

```
'-----[subroutines]-----
```

```
subroutine move forward
```

```
subroutine turnleft to correct position
```

```
subroutine, turn right to correct position
```

```
subroutine, turn left 90 degrees at junction
```

```
subroutine turn right 90 degrees at junction
```

```
subroutine stop position
```

Program 3: Grid Trekking Main Routine PBASIC Program. Refer full program at APPENDIX C-b(ii)

4.3.2.6. Grid Following Program Subroutines

There is addition of two states for grid trekking which are turning right 90 degrees and turning left 90 degrees which are assigned to subroutines 'TurnRightJ' and 'TurnLeftJ' accordingly. When the grid trekker encounters a junction, it will either turn right or turn left 90 degrees or vice versa. This turn is different with turn right and turn left in correction mode. The correction modes are only to maintain the position of the grid trekker on the black grids.

```
'-----[subroutine_TurnRightJ]-----
'subroutine, turn left 90 degrees at junction

TurnRightJ:
'GOSUB GoStop
PAUSE 200
FOR counter = 0 TO 12
    PULSOUT 13, 850      's12 is left
    PULSOUT 12, 850
    PAUSE 90
NEXT
GOSUB GoStop
FOR counter= 0 TO 1
    PULSOUT 13, 650
    PULSOUT 12, 850
NEXT
GOSUB GoStop
RETURN

'-----[subroutine_TurnLeftJ]-----
'subroutine turn right 90 degrees at junction

TurnLeftJ:
PAUSE 200
FOR counter = 0 TO 10    'sb: disabled
    'PAUSE 200
    PULSOUT 13, 650      's13 is right
    PULSOUT 12, 650
    PAUSE 90
NEXT

GOSUB GoStop
FOR counter= 0 TO 2
    PULSOUT 13, 650
    PULSOUT 12, 850
NEXT
GOSUB GoStop
RETURN
```

Program 4: Additional Subroutines in Grid Trekking. Refer full program at APPENDIX C-b(ii)

From 4.2.1.3 *LEDs as Indicators to Verify Input Detection* previously have been mentioned that two subroutines are added to the program to include input detection function to accompany the LED circuit is constructed. These subroutines is called in the main function where '*sensorIndicator*' subroutine detects and lights up when the sensor pairs detects light (or on white region), and '*JuncIndicator*' subroutine lights up a number of LEDs according to the number of junctions detected.

```

DO
    .
    .
    .
    DEBUG CRSRXY, 0, 3, "L= ", BIN1 LDRDetectLeft,
        " R= ", BIN1 LDRDetectRight

GOSUB SensorIndicator
SELECT state
    .
    .
CASE forward
    GOSUB GoForward
    IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
        j = j + 1
        GOSUB JuncIndicator                'led indicator
        IF (j = length(s)) THEN          'we're at end of segment
        HIGH LEDFront
        GOSUB Frontsensor
        state = direction(s)
        j = 0                             'reset junction counter
        HIGH led1
        HIGH led2
        HIGH led3
        HIGH led4
        s = s+1                           'next segment
        ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
            state=tleft
        ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
            state=tright
        ENDIF
        ENDIF
    .
    .
LOOP

```

Figure 32: LEDs Indicators Subroutines Are Called In the Main Routine. Refer full program at APPENDIX C-b(iii)

The subroutines are as follows:

```
'-----[subroutine_JuncIndicator]-----  
JuncIndicator:  
  
    IF( j=0 ) THEN  
        LOW led1  
        LOW led2  
        LOW led3  
        LOW led4  
    ELSEIF( j=1 ) THEN  
        HIGH led1  
        LOW led2  
        LOW led3  
        LOW led4  
    ELSEIF( j=2 ) THEN  
        HIGH led1  
        HIGH led2  
        LOW led3  
        LOW led4  
    ELSEIF( j=3 ) THEN  
        HIGH led1  
        HIGH led2  
        HIGH led3  
        LOW led4  
    ELSEIF( j=4 ) THEN  
        HIGH led1  
        HIGH led2  
        HIGH led3  
        HIGH led4  
    ENDIF  
RETURN  
  
'-----[subroutine_SensorIndicator]-----  
SensorIndicator:  
IF( LDRDetectright=1 ) THEN  
    LOW detectright  
ELSE  
    HIGH detectright  
ENDIF  
  
IF( LDRDetectleft=1 ) THEN  
    LOW detectleft  
ELSE  
    HIGH detectleft
```

Figure 33: Sensor Indicator and Junction Counter LED Indicators Subroutines. Refer full program at APPENDIX C-b(iii)

CONCLUSION

Grid follower robot has focuses on two main parts that are the line sensors and the program. Problems faced during the hardware assembly have been successfully overcome and gives more understanding on the project. Circuit construction needs multiple trying times to finally get the working one. Finite state machine (FSM) is a powerful model for behavior (state) modeling. FSM allows more complex behavior modeling with the aid of state diagram for understanding. Event driven programming enables easier understanding on state machines, respond only to a subset of allowed events and changed directly to only a subset of all possible states. The usage of event-driven programming technique makes programming more comprehensible and neat. Many more cases should be programmed in order to enable to increase the reliability of the grid following program.

REFERENCES

-
- [1] **Nehmzow, Ulrich.** *Mobile Robotics: A Practical Introduction.* s.l. : Springer, 2003.
- [2] **Pakdaman, Mehran and Sanaatiyan, M. Mehdi.** *Design and Implementation of Line Follower Robot.* Second International Conference on Computer and Electrical Engineering, p. 6, 2009.
- [3] **Baharuddin1, M. Zafri, Abidin1, Izham Z. and Mohideen1, S. Sulaiman Kaja.** *Analysis of Line Sensor Configuration for the Advanced Line,* 2006.
- [4] **Parallax Incorporation.** Parallax [Online] <http://www.parallax.com/>.
- [5] **Mazidi, Muhammaad Ali, Mckinlay, Rolind D. and Causey, Danny.** *PIC Microcontroller and Embedded Sytems Using Assembly and C for PIC18.* s.l. : Pearson Prentice Hall, 2008.
- [6] **Jones, Joseph L., Flynn, Anita M. and Seiger, Bruce A.** *Mobile Robots, Inspiration to Implementation, 2nd ed.* s.l. : A K Peters, Ltd, 1999.
- [7] **Ferdinand Wagner, Ruedi Schmuki, Thomas Wagner, Peter Woistenholme.** *Modeling Software with Finite State Machine: A Practical Approach* : Taylor & Francis Group, LLC 2006.
- [8] **Dabek, Frank, et al.** *Event-driven Programming for Robust Software.* MIT Laboratory for Computer Science.
- [9] **Ferdinand AL Williams.** *Microcontroller Projects with Basic Stamps.* : R&D Books, Miller Freeman, Inc. 2000.

APPENDIX A

GANTT CHART

No.	Description/Week	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Building the line sensor													
2	Creating the grids for robot													
3	Programming Boe-Bot with input from sensor													
4	FSM chart and conditions													
5	Writing pseudo code based on FSM chart													
6	Advanced Programming 1 : Programming based on FSM chart													
8	Advanced Programming 2: Use Case Select programming movement													
9	Submission of progress report													
10	Advanced Programming 3: Use Case Select programming movement (several conditions)													
11	Advanced Programming 4: Use Case Select programming; combining various movements in a program.													
12	Pre-EDX (Poster presentation)													
13	Draft report submission													
14	Final report submission													

Table 1: Gantt Chart

APPENDIX B

HARDWARE INFORMATION

a. Parallax Inc. Basic Stamp 2 (BS2) module

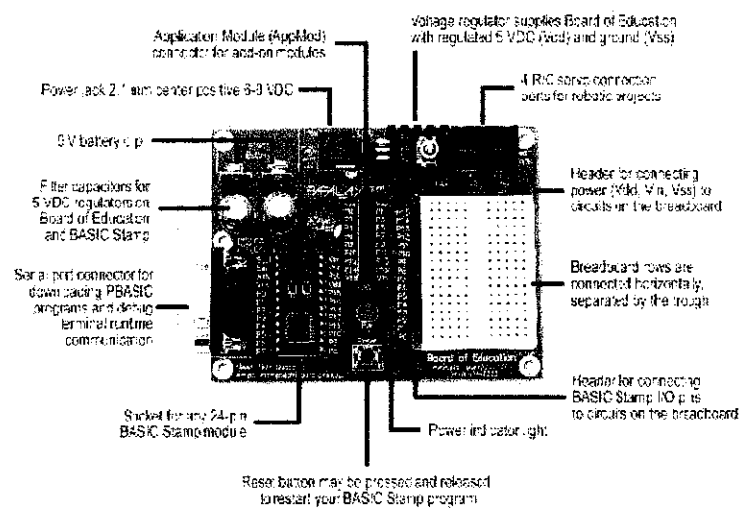


Figure 1: Board Of Education Rev B Carrier Board

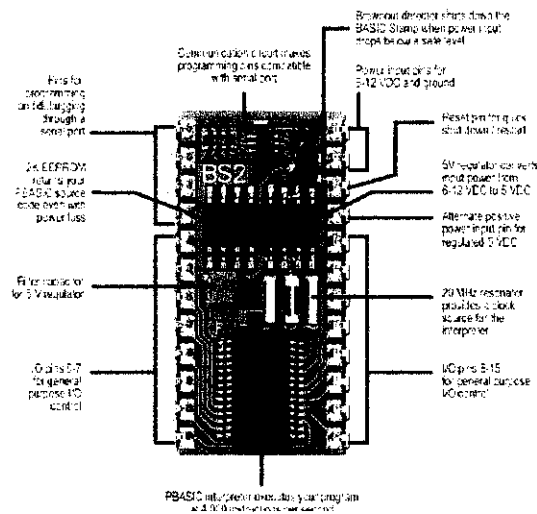


Figure 2: Basic Stamp 2 microcontroller Module

APPENDIX C

PBASIC PROGRAM CODE

Basic movements program

For FYP1, basic movements such as move forward, backward, turn are programmed to the robot. Basic movements can be done simply by controlling the movement of the servo motor. The movements can be clockwise or counter clockwise. These basic movements will be put under subroutines.

a. Basic movement

i. *Pause*

The PAUSE command enables the grid follower robot to stop at some duration of time before executing the next block of codes. We can set the duration of the pause state.

```
'{$STAMP BS2}

'{$PBASIC 2.0}

DEBUG "start timer",CR

PAUSE 1000

DEBUG "one second elapsed...",CR

PAUSE 2000

DEBUG "two seconds elapsed...",CR

DEBUG "Done"

END
```

From the program above, the duration argument is 1000 which equals to 1 second. User can increase the duration by increasing the duration argument between the values of 0-65535.

ii. *Move forward*

Moving forward needs both wheels to turn clockwise. To make the wheel turn clockwise, the duration argument must be less than 750

```
'Robotics with the Boe-Bot - ServoPl3Clockwise.bs2
'Run the servo connected to P13 at full speed clockwise.
'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Program Running!"
DO
PULSOUT 13, 650
PAUSE 20
LOOP

END
```

The coding above enables servo pin 13 to turn clockwise. To make the robot move forward, both wheels must turn clockwise as the program below:

```
'Robotics with the Boe-Bot - MoveForward.bs2
'Run the servo connected to P13 at full speed clockwise.
'{$STAMP BS2}
'{$PBASIC 2.5}

DEBUG "Program Running!"
DO
PULSOUT 13, 650
PULSOUT 12, 650
PAUSE 20
LOOP

ENDS
```


iii. *Move Backward*

Moving backward needs both wheels to turn counter clockwise. To make the wheel turn clockwise, the duration argument must be more than 750. Ideal duration for full speed rotation is 850.

```
' Robotics with the Boe-Bot - ServoP13Clockwise.bs2
' Run the servo connected to P13 at full speed clockwise.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"
DO
PULSOUT 13, 850
PAUSE 20
LOOP

END
```

The coding above enables servo pin 13 to turn clockwise. To make the robot move forward, both wheels must turn clockwise as the program below:

```
' Run the servo connected to P13 at full speed clockwise.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG "Program Running!"
DO
PULSOUT 13, 850
PULSOUT 12, 850
PAUSE 20
LOOP

END
```

iv. Turn

To make the robot turn, both wheels must turn in opposite direction. Based on the coding below, servo pin 13 is turning counter clockwise and servo pin 12 is turning clockwise.

```
' Robotics with the Boe-Bot - ServosP13CcwP12Cw.bs2
' Run the servo connected to P13 at full speed
counterclockwise
' and the servo connected to P12 at full speed clockwise.

' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"
DO
PULSOUT 13, 850
PULSOUT 12, 650
PAUSE 20
LOOP
```

v. Pivot

To make the robot turn, one wheels must stop and another one turn either clockwise or counter clockwise. Based on the coding below, servo pin 13 is turning counter clockwise and servo pin 12 is in stopping mode.

```
' {$STAMP BS2}
' {$PBASIC 2.5}
DEBUG "Program Running!"
DO
PULSOUT 13, 850
PULSOUT 12, 750
PAUSE 20
LOOP
```

b. Full Programs

For FYP2, more advanced programming is done, requiring the Boe Bot maneuver based on the input sent by the sensor. Embedded with sensor, the Grid Trekker robot can perform line trekking function. This means this Grid Trekker robot can now differentiate bright and dark surfaces, allowing it to follow a line.

The next stage of programming is to program the robot to follow grids instead of just lines. This Grid Trekker robot must maneuver based on the program and follows certain route that has been programmed to it.

i. Line trekking from sensor input (conventional programming)

```
' -----[ Title ]-----
' Useful for following a 2.25 inch wide vinyl electrical tape
stripe.
' {$STAMP BS2} ' Stamp directive.
' {$PBASIC 2.5} ' PBASIC directive.
' -----[ Constants ]-----
LEDleft      CON  10                      'led is active
high
LEDright     CON  0

' -----[ Variables ]-----
LDRDetectLeft  VAR  Bit
LDRDetectRight VAR  Bit
intersect      VAR  Byte
' -----[ Main Routine ]-----

DEBUG "Program Running!"
HIGH LEDright      'led on
HIGH LEDleft

DO
  LDRDetectLeft = IN3      'left led is assigned to i/o pin 3
  LDRDetectRight = IN6     'left led is assigned to i/o pin 6

  DEBUG CR$RXY, 0, 3, "L = ", BIN1 LDRDetectLeft,
    "      R = ", BIN1 LDRDetectRight
```

```

IF (LDRDetectLeft = 1) THEN
    IF (LDRDetectRight = 1) THEN                ' L = 1, R = 1
        GOSUB GoForward
    ELSE                                          ' L = 1, R = 0
        GOSUB TurnRight
    ENDIF
ELSE
    IF (LDRDetectright = 1) THEN                ' L = 0, R = 1
        GOSUB TurnLeft
    ELSE
        GOSUB GoStop
    ENDIF

ENDIF
PAUSE 20

LOOP
' -----[ Subroutine - GoForward ]-----
GoForward:
    PAUSE 20
    PULSOUT 13, 650
    PULSOUT 12, 850
    RETURN
'-----[subroutine_idlez]-----
idlez:
IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
'FOR counter = 1 TO 120
PULSOUT 13, 750
PULSOUT 12, 750
PAUSE 20
'NEXT
ENDIF
RETURN
'-----[subroutine_GoStop]-----
GoStop:
FOR counter = 1 TO 8
PAUSE 20
PULSOUT 13, 750
PULSOUT 12, 750
PAUSE 20
NEXT
RETURN

```

```

'-----[subroutine_GoForward]-----
GoForward:
PAUSE 20
'FOR counter = 1 TO 4
PULSOUT 13, 650
PULSOUT 12, 850
PAUSE 20
'GOSUB GoStop
'NEXT
RETURN
'-----[subroutine_TurnRight]-----
TurnRight:
PAUSE 20
FOR counter = 0 TO 1
PULSOUT 13, 850          's12 is left
PULSOUT 12, 850
PAUSE 20
GOSUB GoStop
NEXT
RETURN
'-----[subroutine_TurnLeft]-----
TurnLeft:
PAUSE 20
FOR counter = 0 TO 1
PULSOUT 13, 650          's13 is right
PULSOUT 12, 650
PAUSE 20
GOSUB GoStop
NEXT
RETURN

```

ii. *Line trekking from sensor input (CASE...SELECT)*

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'-----[constants]-----
idle      CON 0
pauze     CON 1
forward   CON 2
tright    CON 3
tleft     CON 9

LEDRight  CON 10
LEDLeft   CON 0
'-----[variables]-----
state      VAR Byte
LDRDetectRight VAR Bit
LDRDetectLeft  VAR Bit
counter      VAR Byte
intersect     VAR Byte
'-----[main routine]-----
DEBUG CR,"program running!"

HIGH LEDright
HIGH LEDleft

DO
LDRDetectright = IN3      'assign input to pin 3 and pin 6
LDRDetectleft  = IN6

'compare input with crsrxy
DEBUG CRSRXY, 0, 3, "L= ", BIN1 LDRDetectLeft,
" R= ", BIN1 LDRDetectRight

IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
state=forward
ENDIF

CASE forward
GOSUB GoForward
IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
state=pauze
ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
state=tleft
ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
state=tright
ENDIF
```

```

CASE pauze
GOSUB GoStop
IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
state=tleft
ENDIF

CASE tleft
'GOSUB TurnLeft
GOSUB PivotLeft
IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
state=forward
ENDIF

CASE tright
'GOSUB TurnRight
GOSUB PivotRight
IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
state=forward
ENDIF

ENDSELECT
PAUSE 20

LOOP

'-----[subroutine_idlez]-----
idlez:
IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
'FOR counter = 1 TO 120
PULSOUT 13, 750
PULSOUT 12, 750
PAUSE 20
'NEXT
ENDIF
RETURN
'-----[subroutine_GoStop]-----
GoStop:
FOR counter = 1 TO 8
PAUSE 20
PULSOUT 13, 750
PULSOUT 12, 750
PAUSE 20
NEXT
RETURN

```

```

'-----[subroutine_GoForward]-----
GoForward:
PAUSE 20
'FOR counter = 1 TO 4
PULSOUT 13, 650
PULSOUT 12, 850
PAUSE 20
'GOSUB GoStop
'NEXT
RETURN
'-----[subroutine_TurnRight]-----
TurnRight:
PAUSE 20
FOR counter = 0 TO 1
PULSOUT 13, 850          's12 is left
PULSOUT 12, 850
PAUSE 20
GOSUB GoStop
NEXT
RETURN
'-----[subroutine_TurnLeft]-----
TurnLeft:
PAUSE 20
FOR counter = 0 TO 1
PULSOUT 13, 650          's13 is right
PULSOUT 12, 650
PAUSE 20
GOSUB GoStop
NEXT
RETURN

```


iii. *Grid trekking from sensor input with LED indicator (CASE...SELECT)*

```
' {$STAMP BS2}
' {$PBASIC 2.5}
'line trekking coding 8
'-----[constants]-----
idle          CON 0
pauze         CON 1
forward       CON 2
trightj       CON 12
tleftj        CON 13
tright        CON 3
tleft         CON 4
setstop       CON 2
setstop2      CON 2

LEDRight      CON 14      'at sensor
LEDLeft       CON 15      'at sensor
detectleft    CON 2
detectright   CON 0
led1          CON 7        'indicator for junction, j=1
led2          CON 9        'indicator for junction, j=2
led3          CON 11       'indicator for junction, j=3
led4          CON 10       'indicator for junction, j=4
'-----[variables]-----
state         VAR  Byte
LDRDetectRight VAR  Bit
LDRDetectLeft VAR  Bit
counter       VAR  Byte

'arrays to store distance and direction of turn
MAXSEG        CON 5
length        VAR  Byte (MAXSEG)
lengthB       VAR  Byte (MAXSEG)
direction     VAR  Byte (MAXSEG)
directionB    VAR  Byte (MAXSEG)
j             VAR  Byte
s             VAR  Byte
seg           VAR  Byte
```

```

'-----[main routine]-----
DEBUG CR,"program running!"
  HIGH LEDright
  HIGH LEDleft
'----- define route -----
length(0) = 2
length(1) = 3
length(2) = 2
length(3) = 4
length(4) = 2
direction(0) = trightj
direction(1) = tleftj
direction(2) = trightj
direction(3) = tleftj
direction(4) = pauze
seg = 4           'number of total segments in this route
j = 0             'reset junction counter
s = 0             'start at first segment
state = 0         'added to initialize state to zero

'define inputs
INPUT 3
INPUT 6

'Preset LEDs state
LOW led1
LOW led2
LOW led3
LOW led4

DO

  LDRDetectright = IN3
  LDRDetectleft = IN6

  DEBUG CRSRXY, 0, 3, "L= ", BIN1 LDRDetectLeft,
    " R= ", BIN1 LDRDetectRight

GOSUB SensorIndicator

SELECT state

CASE idle
  IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

```

```

CASE forward
  GOSUB GoForward
  IF (LDRDetectLeft=0) AND (LDRDetectRight=0) THEN
    j = j + 1          'increment junction counter
  GOSUB JuncIndicator  'led indicator

  IF (j = length(s)) THEN      'we're at end of segment
    HIGH LEDFront
    GOSUB Frontsensor
    state = direction(s)
    j = 0                    'reset junction counter

    HIGH led1
    HIGH led2
    HIGH led3
    HIGH led4
    s = s+1                  'next segment

    ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
      state=tleft
    ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
      state=tright
    ENDIF
  ENDIF

CASE tleft
  GOSUB TurnLeft
  IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

CASE tright
  GOSUB TurnRight
  IF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

CASE tleftj
  GOSUB TurnLeftJ
  IF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
    state=tright
  ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
    state=tleft
  ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
    state=forward
  ENDIF

```

```

CASE trightj
    'GOSUB TurnLeftJ
    GOSUB TurnRightJ
    IF (LDRDetectLeft=1) AND (LDRDetectRight=0) THEN
        state=tright
    ELSEIF (LDRDetectLeft=0) AND (LDRDetectRight=1) THEN
        state=tleft
    ELSEIF (LDRDetectLeft=1) AND (LDRDetectRight=1) THEN
        state=forward
    ENDIF

CASE pauze
    EXIT

ENDSELECT
PAUSE 20

LOOP

    LOW LEDright
    LOW LEDleft
    DEBUG CR,"The End."

END

'-----[subroutine_GoForward]-----
GoForward:
    FOR counter = 0 TO 10
        PULSOUT 13, 740
        PULSOUT 12, 760
        PAUSE 15
        'GOSUB GoStop
    NEXT
RETURN
'-----[subroutine_TurnRightJ]-----
TurnRightJ:
    PAUSE 200
    FOR counter = 0 TO 12
        PULSOUT 13, 850          's12 is left
        PULSOUT 12, 850
        PAUSE 90
    NEXT

    GOSUB GoStop
    FOR counter= 0 TO 1
        PULSOUT 13, 650
        PULSOUT 12, 850
    NEXT
    GOSUB GoStop
    RETURN

```

```

'-----[subroutine_TurnLeftJ]-----
TurnLeftJ:
'GOSUB GoStop
PAUSE 200
FOR counter = 0 TO 10      'sb: disabled
    'PAUSE 200
    PULSOUT 13, 650        's13 is right
    PULSOUT 12, 650
    PAUSE 90
    'GOSUB GoStop
NEXT                        'sb: disabled

GOSUB GoStop
FOR counter= 0 TO 2
    PULSOUT 13, 650
    PULSOUT 12, 850
NEXT
GOSUB GoStop
RETURN
'-----[subroutine_TurnRight]-----
TurnRight:
'FOR counter = 0 TO 1      'sb: disabled
    PULSOUT 13, 770        's12 is left
    PULSOUT 12, 770
    PAUSE 50
    GOSUB GoStop
    ' NEXT                  'sb: disabled
RETURN
'-----[subroutine_TurnLeft]-----
TurnLeft:
' FOR counter = 0 TO 1      'sb: disabled
    PULSOUT 13, 730        's13 is right
    PULSOUT 12, 730
    PAUSE 50
    GOSUB GoStop
    ' NEXT                  'sb: disabled
RETURN
'-----[subroutine_GoStop]-----
GoStop:
'FOR counter = 0 TO 1
    PAUSE 30
    PULSOUT 13, 750
    PULSOUT 12, 750
    PAUSE 30
'NEXT
RETURN

```

```

'-----[subroutine_JuncIndicator] -----
JuncIndicator:
      IF( j=0 ) THEN                                'sb:added 'to reset leds
when j=0      LOW  led1                                'sb:added
              LOW  led2                                'sb:added
              LOW  led3                                'sb:added
              LOW  led4                                'sb:added
      ELSEIF( j=1 ) THEN                            'sb:modified from if ->
elseif
              HIGH led1
              'LOW led1
              LOW  led2                                'sb:added
              LOW  led3                                'sb:added
              LOW  led4                                'sb:added
      ELSEIF( j=2 ) THEN
              HIGH led1
              HIGH led2
              'LOW led1
              'LOW led2
              LOW  led3                                'sb:added
              LOW  led4                                'sb:added
      ELSEIF( j=3 ) THEN
              HIGH led1
              HIGH led2
              HIGH led3
              'LOW led1
              'LOW led2
              'LOW led3
              LOW  led4                                'sb:added
      ELSEIF( j=4 ) THEN
              HIGH led1
              HIGH led2
              HIGH led3
              HIGH led4
              'LOW led1
              'LOW led2
              'LOW led3
              'LOW led4
      ENDIF
RETURN

```

```

'-----[subroutine_SensorIndicator]-----
SensorIndicator:
  IF( LDRDetectright=1 ) THEN          'LED indicator when right
sensor detects white region
    LOW detectright
  ELSE
    HIGH detectright
  ENDIF

  IF( LDRDetectleft=1 ) THEN
    LOW detectleft
  ELSE
    HIGH detectleft
  ENDIF

  IF( LDRDetectFront=1 ) THEN
    LOW detectfront
  ELSE
    HIGH detectfront
  ENDIF
RETURN
'-----[subroutine_Frontsensor]-----
Frontsensor:
IF (LDRDetectFront = 0) THEN
  IF (j = length(s)) THEN
    HIGH LEDFront
    state = directionB(s)
    j = 0
    s = s+1
  ENDIF
ENDIF
ENDIF

```