Lego NXT: Exploring Real World Interactive Systems

Christos Liontos

A report submitted as part of the requirements
for the degree of MSc in Computing: Software Technology
at the School of Computing Science and Digital Media
Robert Gordon University, Aberdeen, Scotland

August 2014

Supervisor Dr. Daniel C Doolan

Abstract

Lego Mindstorms NXT is a robotics kit developed by Lego industry in July 2006. It contains a programmable brick, the brain of the robot, along with a great variety of sensors that allow to design and program robots using various programming languages. Since the day it was released, it has been mostly popular in educational institutions and has been used as a tool to introduce the basics in robotics development. However, its potentials were soon obvious, attracting even professional programmers into designing complicated projects that were about to make NXT robots a lot more popular.

This project was designed to explore the NXT capabilities. Its purpose is to create an interactive environment where humans will be able to communicate with the robot and command it to perform specific tasks. The interaction is accomplished through the use of a Java Graphical User Interface application. The robot operates in a limited environment and builds patterns according to the user's instructions. The project is divided in two fundamental parts, the Interface application and the Robot program development.

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr. Daniel C. Doolan for his thorough support throughout the investigation and implementation of the project. His continued guidance helped me to complete this task through various stages.

I also take the opportunity to express a deep sense of gratitude to Dr. Jean Claude Golovine for his valuable lectures in Object Oriented Programming that formed my main inspiration for the Interface development.

Declaration

I confirm that the work contained in this MSc project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Signed ..... Date .....

Christos Liontos

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The following thesis aims to investigate the capabilities of Lego NXT robots. The objective is to explore the NXT potentials by designing an autonomous robot, able to interact with a user via the computer and perform a number of tasks. It is about a combination of Graphical User Interface design and Robot Programming, using the JAVA programming language. A Three month prior research had been conducted before proceeding to the second stage of the actual project. Many factors have been taken under consideration in order to decide about the main functionality such as the robot performance, objectives and sensors to be used.

1.1 Background

The project is about an autonomous builder robot operating inside a frame. It aims to design a fork-lift robot able to build specific patterns over a $10X10$ Grid, under the user's instructions. Using different types of sensors will be essential. The user will be able to select various target locations (squares) where the robot will build on (Figure 1.1). The project's functionality is divided into two main parts. The first part is related to the Graphical User Interface while the second one to the actual Robot. The GUI plays the role of the interpreter between the human and the robot. It should translate the user's input into robotic commands. The main objectives of the project are mentioned below. In addition, the picture following offers a better understanding of the project's overview.

- Design a GUI that allows the user communicate with the robot.
- Achieve communication between the GUI and the NXT brick.
- Design a robotic model capable of performing the desired tasks.
- Program the robot to build the desired patterns using its sensors to navigate.

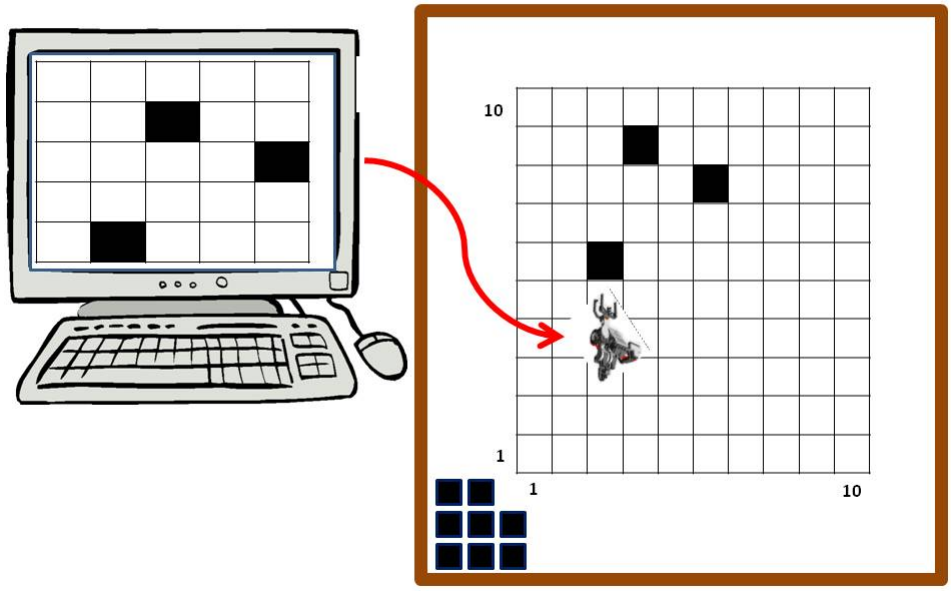


Figure 1.1: Project overview

Below is a brief recap of the investigation stage highlighting the main findings that impacted many of the design and implementation decisions.

1.1.1 NXT Sensors and Additional Parts

Research was conducted to identify the NXT robot's potentials such as the available hardware (sensors, parts) to be used for the project. Each part and sensor was examined separately offering a clear view of the robot's specifications. Finally a small introduction was made to the robot's brain, the brick.

Conclusions: There is a wide range of sensors to be used for the projects offering plenty of alternatives. Additionally, the huge variety of Lego parts makes it easy to create any kind of robotic model.

1.1.2 Research on Already Existed Projects

Research had to be done regarding already existed projects. Many cases were discovered that enhanced the research and made the NXT's capabilities even more clear. There was no doubt anymore that the robot had a wide range of potentials.

Conclusions: No specific conclusions. However, many ideas were gained during this research. Such ideas were the use of a mechanism to pick up the bricks as well as the line following ability that makes a movement more accurate.

1.1.3 Communications

The communication research was one of the major fields during the project investigation. Computer interaction with the robot is essential, thus we should be certain that the Graphical User Interface can achieve a connection with the brick. A brief introduction to Bluetooth technologies was made as well.

Conclusions: The robot can communicate with a JAVA Graphical User Interface either via USB or Bluetooth technologies. Both ways should be implemented to the project.

1.1.4 Discussion and Evaluation of Software Tools to be Used

There are several software options to be used for programming an NXT robot. Research was conducted to identify the best programming language and any additional software required. The most common IDEs were examined and advantages and disadvantages were recorded. In addition, attempt was made to install some of these for a better view.

Conclusions: Java programming language was decided to be used Thus, eclipse IDE was the most proper.

1.1.5 Primary Research: Exploratory Code Development

During this phase, research was made on various code samples found on the web. An attempt was made to create our own code and experiment with the sensors and the brick. Small projects were designed to familiarize ourselves with robotic programming. By the end of this research most sensors had been programmed to perform the most basic activities and their main methods had been identified.

Conclusions: Experience was gained regarding programming the robot

1.1.6 Problems Identified and Proposed Solutions

This section summarized the basic problems identified during the experiments with programming the robot. It contained the basic conclusions and results of the tests conducted during the project investigation.

Conclusions: The major problems identified were related to robot rotations, robot navigation and programming with threads.

1.2 About this Thesis

This is the thesis of *Christos Liontos*, submitted as part of the requirements for the degree of MSc Computing: Software Technology at the School of Computing, Robert Gordon University, Scotland.

1.3 Chapter List

Chapter 2 Design. This part of the report deals with the prototype designs and initial ideas of the project.

Chapter 3 Implementation. This chapter aims to describe the solutions given to fulfill the objectives. Contains information about the GUI design, the robot mechanical design and the actual programming techniques used.

Chapter 4 Evaluation & Testing. Includes the tests conducted to evaluate the final version of the project.

Chapter 5 Conclusion. The conclusions of the thesis are presented.

1.4 Conclusion

The project had to be divided in smaller parts. Each one of them should be examined and tested separately before combined with others to form the final result. The following chapters go through the entire process of designing and implementing the project. Each section examines a new attribute and explains any difficulties encountered and how they were overcome. Second and third chapters form the main body of the project. They describe both the prototype designs and the implementation procedure. Finally, testing and conclusions can be found in the last chapters.

Chapter 2

Design

The chapter bellow summarizes the prototype designs of the project. Those are the initial ideas and plans to achieve the basic objectives. However, some of them were abandoned after showing up either unreliable or too complicated to be implemented. In addition, more effective alternatives were found after running several tests during the implementation phase.

2.1 Prototype Project Design

Figure 2.1 indicates the conception of the initial idea regarding the cooperation between the computer and the Graphical User Interface. There had been no attempts to design any problem solving algorithms at that time. The robot was supposed to move freely inside the frame using both Ultrasound and Compass navigation techniques. The idea of using the compass was later discarded as it was figured out that the one and only available sensor did not operate properly.

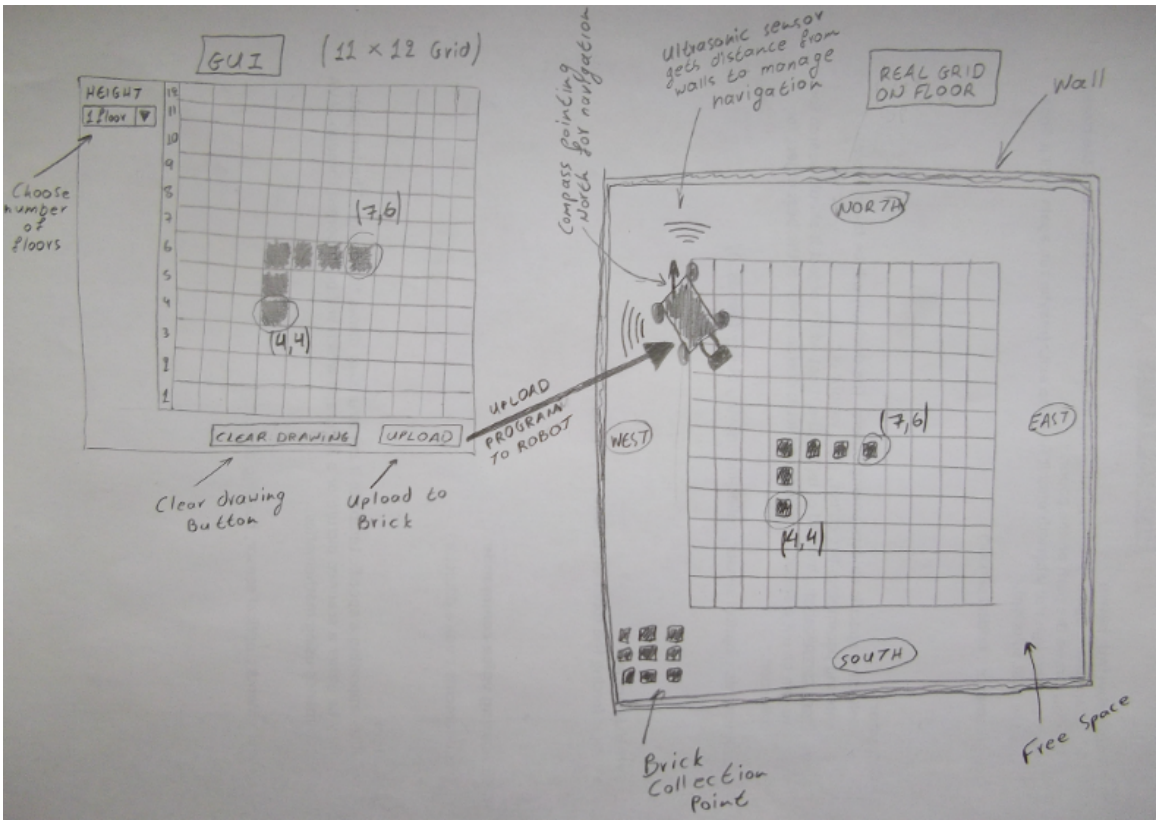


Figure 2.1: The GUI (to the left), the Frame (to the right)

Figures 2.2,2.3 depict the first pseudocodes designed. Some major problems were identified and solutions were proposed. Those sketches indicate the first attempts made to record the most significant events to occur during the procedure. They form the basic plans followed during the implementation phase. The following conclusions were made:

1.The user should be able to select squares randomly.

2.The GUI should be capable to communicate with the robot and send it the selected square's coordinates.

3.The robot should be capable of translating these coordinates into real distances.

4.The robot should be able to decide which square should be visited first to avoid blocking the way for the next ones. This can be achieved by sorting the coordinates using the Bubble-sort method.

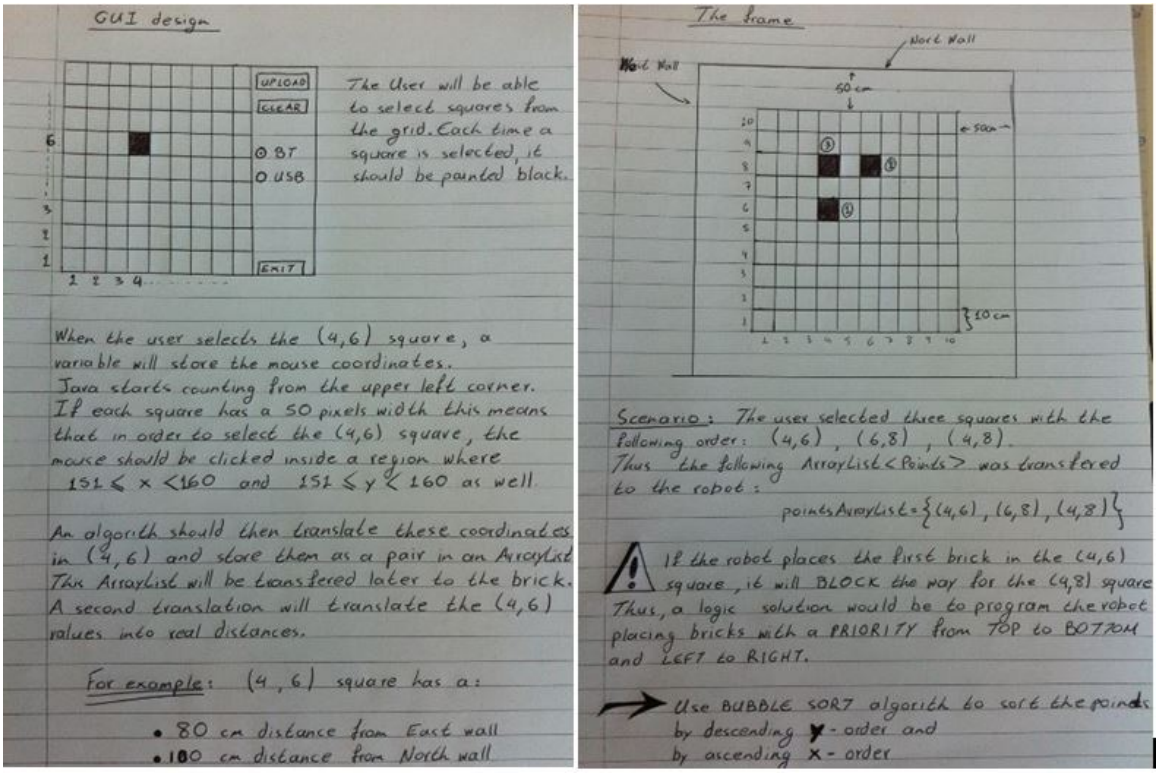


Figure 2.2: GUI - ROBOT interaction

The last pair of pictures depicts the prototype designs of the robot. They refer to the mechanical gears to be used such as the operator and the movement system. The basic advantages and disadvantages were identified. As a result, the following decisions were made:

1.The claw operator is not fit for purpose. The desired functionality is the robot to be capable of lifting the brick from above, without disturbing the nearby ones. Thus, the fork-lift option will be used.

2.While attaching two motors with four wheels, movement should pass from the two motors to all four wheels. This means extra gears should be used (to connect front with back wheels) that require a lot of space. In addition, turning will be a much more complicated procedure. Thus, two wheels will be attached to the motors. A third separate wheel will be used to maintain stability.

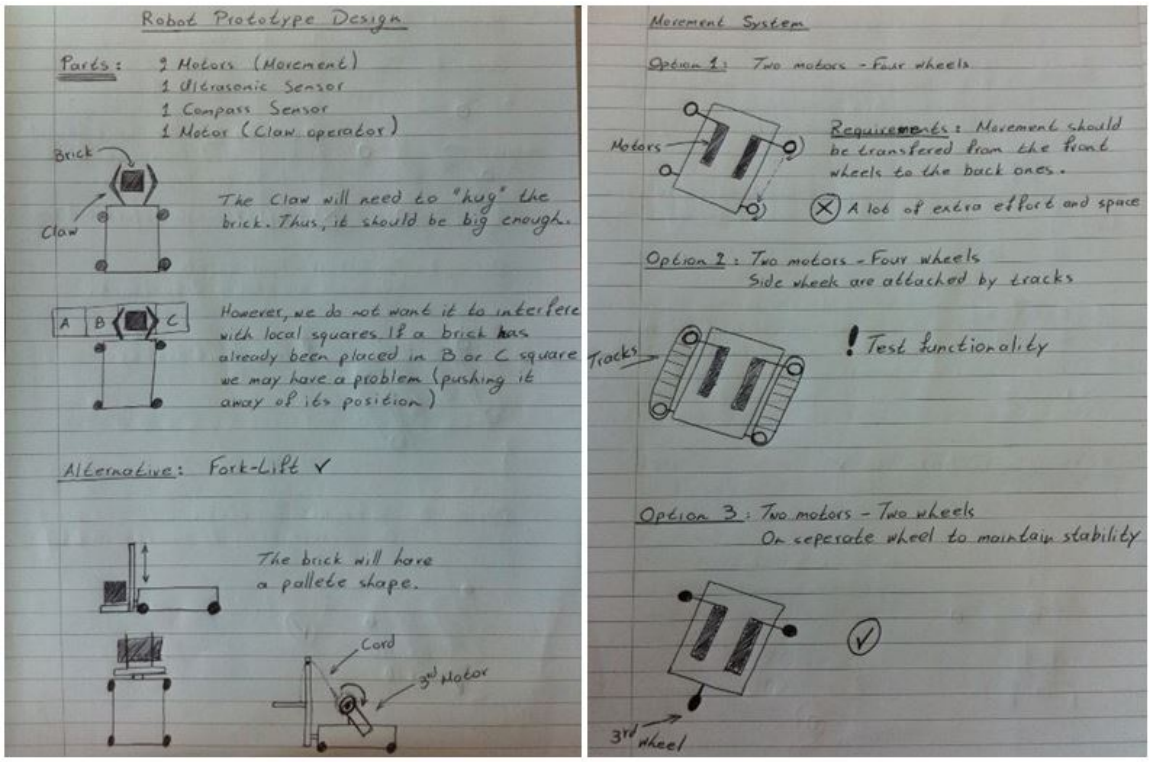


Figure 2.3: Prototype Mechanical design

2.2 Overview of robot navigation

As mentioned before, the project is about a builder robot. It is quite obvious that an Interface is essential to achieve communication between the user and the robot. By the time the robot gets the proper instructions by the user it should initiate the procedure to construct the designed layout. A number of sensors are used for navigation. Their main responsibility is to inform the robot of its current position and allow it track specific points of interest such as the desired squares.

During the implementation phase, the prototype design used one Compass and one Ultrasonic sensor for navigating inside the frame. The Ultrasonic sensor was used to measure the distance from the walls while the Compass sensor to maintain a direction. In more details, it would lock to a specific angle, the one towards the North wall, and maintain it while moving forward. In other words, it would make the robot capable of moving parallel to the West wall. However, after conducting several tests it was discovered that the compass sensor did not function properly. The reading values were inaccurate leading to the conclusion that the sensor was not fit for purpose. Thus, alternatives had to be found to achieve the desired tasks.

The last version is based on the use of two Ultrasonic sensors. One of them faces Backwards while the other to the Left side. Such formation, make them capable of recording the back and side distances from the walls (West and South wall) at the same time. The side Ultrasonic sensor has replaced the Compass sensor. It continuously tracks the distance from the West wall. When the distance changes the robot steers left or right to correct it. The result is the same like using the Compass to lock on a specific angle. The robot can maintain a straight route, allowing it to travel parallel to the West wall. In addition, a color sensor is used in finding the dispatch position. It provides a more accurate movement towards the brick dispenser, demonstrating the power of the color recognition. Before diving into further details, the main objectives should become more clear. Figure 2.4 depicts the actual grid and the frame inside which the robot will operate. Further below, pseudo-code 1 informs about the order the robot follows to perform its actions after receiving the instructions.

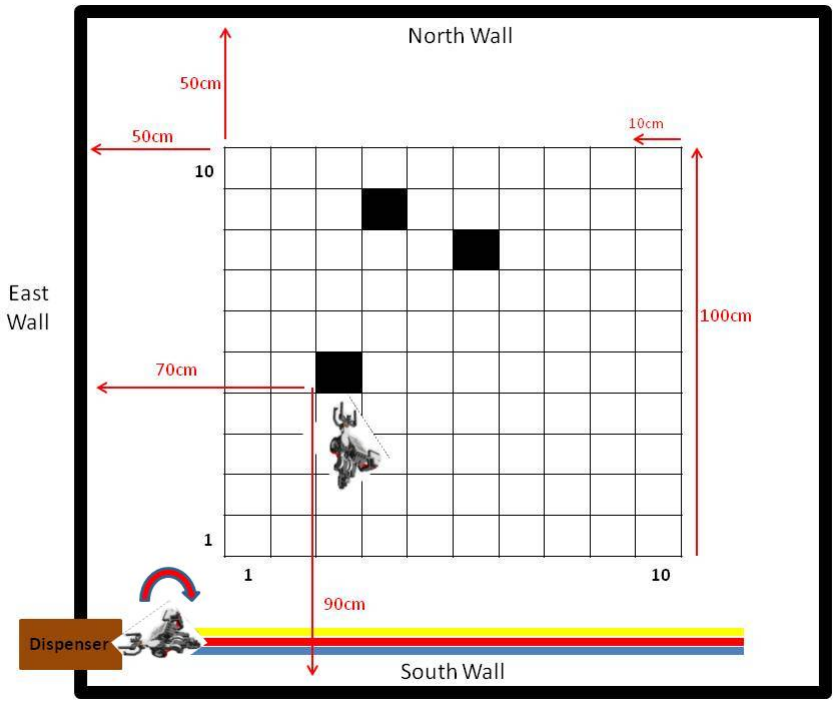


Figure 2.4: The procedure

Algorithm 1: High Level Operation Events

Get coordinates of the first square to be placed ((3, 5) etc.);
Pick up the first brick;
Rotate 180 degrees;
Move forward until distance from East wall = 50 + (x-coordinate). While moving, use the color sensor to follow the line, maintaining a straight route;
Rotate 90 degrees and face to the North wall;
while *distance from the South wall < 50 + (y-coordinate)* **do**
    Move Forward; **while** *Moving* **do**
        Use the rear Ultrasonic sensor to maintain the x-coordinate. This will keep it in a straight route;
    end
end
Place brick on the square;
Move backward until the color sensor senses the line;
Rotate 90 degrees and face to the West; **while** *Away from dispatch position (Not blue color is found)* **do**
    Move Forward; **while** *Moving* **do**
        Use the color sensor to follow the line, maintaining a straight route;
    end
end
Stop;
Initiate the same procedure for the next square to be visited;

Taking under consideration the above events, the main functionality was discovered and broken into pieces. It is quite obvious that the most important parts of the procedure will be the following:

1. The GUI and the robot should communicate efficiently.
2. The robot should be able to lift bricks,
3. Follow lines,
4. Measure distances,
5. Move parallel to the East wall,
6. Execute the same procedure for various squares.

The following sections inform about the technical difficulties discovered during the implementation of the project and the solutions provided. Each problem was examined separately. The binding of the parts came later, after every problem had been solved.

2.3 Robot Mechanical Design

Working with robots is not just pure programming. It also includes mechanics. Motors, Robotic arms, Claws and Sensors are all part of the robot long before programming begins. A good model that can take advantage of all its parts without generating inconsistencies may lead to much less programming effort. Before building a robot, one should think about the desired functionality. It is this that will determine the specific design. The model to be used for this project has to satisfy the following requirements:

- Have small size to move easily inside the limited space.
- Perform forward and backward movements.
- Perform rotations (probably 90 degree).
- Sense the distances between itself and the walls.
- Recognize specific colors on the floor.
- Move objects inside the frame and arrange them in specific positions.
- Overcome small obstacles, like tapes or scratches, on the floor.

2.3.1 Managing the design

The first challenge would be to design a small sized robot able to perform a lot of functions. LEGO has a huge variety of parts to be used for building various types robots. However, sometimes it is quite impossible to fit many sensors and motors all together. Having a look at the requirements mentioned above and according to the project's objectives the following items would be essential:

1. At least two motors for movement.
2. At least two sensors (Ultrasonic/Compass and Color sensor) for navigation.
3. A third motor to operate the Claw for picking up the objects.
4. The brick, which needs a lot of space.

2.3.2 The movement system

When designing a Lego robot, everything starts with the motors. There is no alternative if the model needs to achieve movement. The motors should be placed on the lower part of the robot. However, they can be used in many different ways. One has to

decide if he is going to use two wheels, four wheels, tracks etc. Four wheels may offer better performance but they are difficult to install since movement from the two motors must be directed to four wheels. In addition, they need much more space than two wheels. Tracks on the other hand are capable of performing with accuracy in any kind of surface, but they are slow and do not fit very well to the motors. Sometimes, there is a lot of free space inside the tracks which make them jump of the wheels. Two wheels may use much less space, but they lack of stability, thus additional parts have to be added which will probably be separate from the wheels. Figure 2.5 depict some models used in various projects.

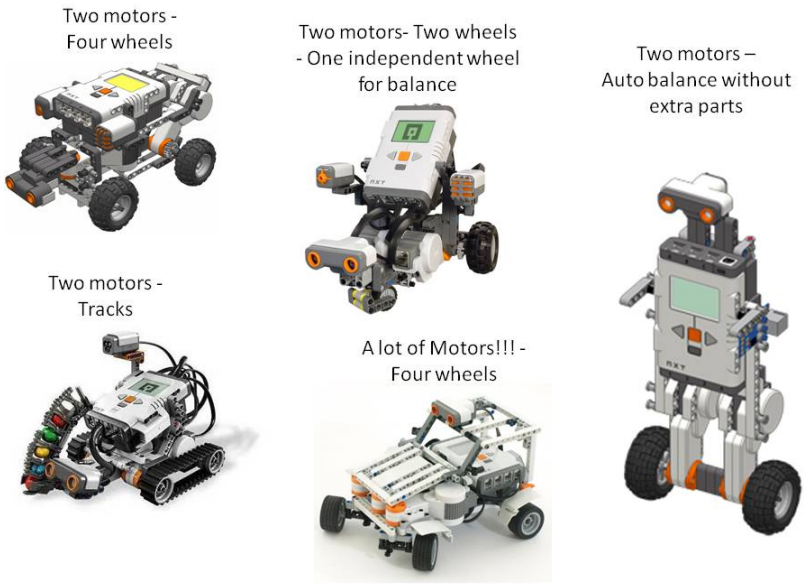


Figure 2.5: Various movement designs

Many combinations of the above models have been tried while searching for the best solution. As a conclusion, using two wheels in collaboration with a separate third one seemed to be the most appropriate. The two motors were placed horizontally side by side to each other. A third wheel followed behind to maintain stability. The advantages were limited size and a lot of free space left above the wheels for the brick and the sensors (Figure 2.6).

Figure 2.6: Initial Movement system. One balancing wheel used

However, after several tests on different surfaces, it became obvious that the third wheel could not operate efficiently. It was only able to move backward and forward generating problems when the robot had to rotate, as it should literally drag the third wheel to the side. Even a slight obstacle (such as tape) could immobilize the vehicle if the third wheel was dragged onto it. In order for this problem to be solved, the wheel was replaced by two smaller ones located to the rear, left and right side of the robot (Figure 2.7). The wheels can still maintain stability successfully and also rotate 360 degrees independently from each other. The latest design reduced significantly the possibility of having the robot stuck to a random obstacle.

Figure 2.7: Second version of Movement system. Two balancing wheels. Better performance on avoiding small obstacles

2.3.3 The Fork-Lift

Another important objective for the project is the robot to be able to move objects around the frame. This means that a third motor should be used to operate a specific mechanism. The initial idea was to use a mechanical claw that could grab an object and hold it still. However, this meant that the claw should 'hug' it all around. Thus, it should be quite big since the bricks themselves are wide. In addition, the robot would

drag the bricks on the floor which could lead to several problems in case they got stuck somewhere. And since tapes are going to be used on the floor this would be dangerous. An alternative solution was to use a fork lift. A fork-lift can lift an object from the bottom instead of grabbing it from the sides. This is very useful since it will never disturb other bricks that may be located in a position right next to the current one. In addition, the brick is lifted in the air which means that by no chance it can interfere with the floor. By the time a brick is lifted one should worry only for the motors to avoid any obstacles. And despite the big shape of the fork-lift, it does not add more size to the robot since it extends vertically. The motor to operate the fork-lift was placed between of the two wheel motors leaving a lot of free space for the brick (Figure 2.8).

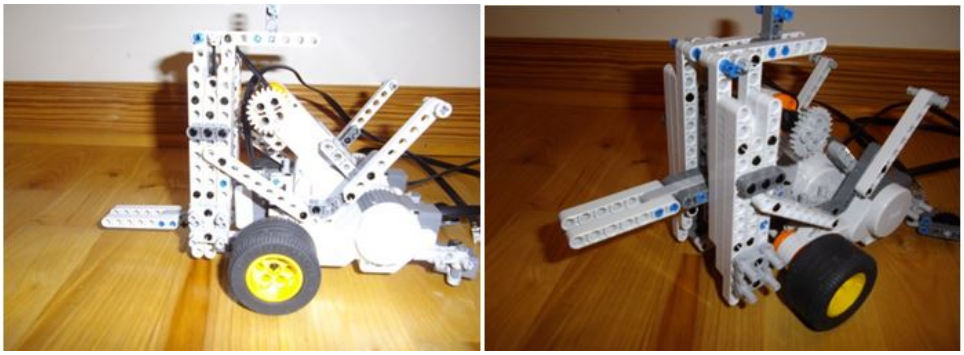


Figure 2.8: Fork-lift operator

2.3.4 The sensors and the brick

Once the motors and the lift were fit together, it was not much of a challenge to place the sensors. Two Ultrasonic sensors were placed on the upper side of the lift. One of them is facing forward while the other one to the left. This way the robot will be able to record the distance between the walls and its two sides. A color sensor was placed between the two wheel motors. It faces downwards and is capable of recognizing various colors on the floor. It will be used to recognize stripes of tape that will lead to the dispatch point. Finally, the brick was placed on the back side of the robot, above the wheels. It offers a nice balance to the model and does not consume a lot of useful space (Figure 2.9).

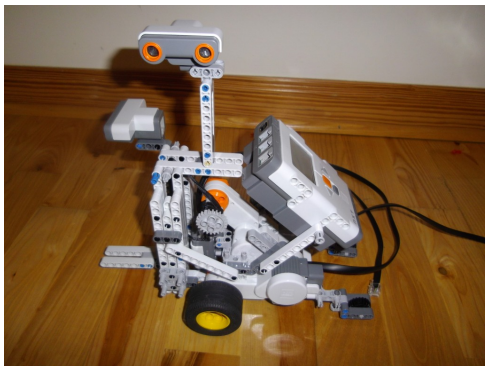


Figure 2.9: Prototype robot design. Combines all the desired Sensors and Motors

2.3.5 Lego Digital Designer (LDD)

Lego Digital Designer (Figure 2.10) is a Software created by Lego that allows users to build models using virtual bricks. By the time the design is completed the program has the ability to record every part used for the specific model and generate step by step instructions. It is a very useful tool that offers the opportunity to share a project with people who may be interested in creating the same, or similar models in the future. The software is free and can be downloaded by the official site of Lego [1].

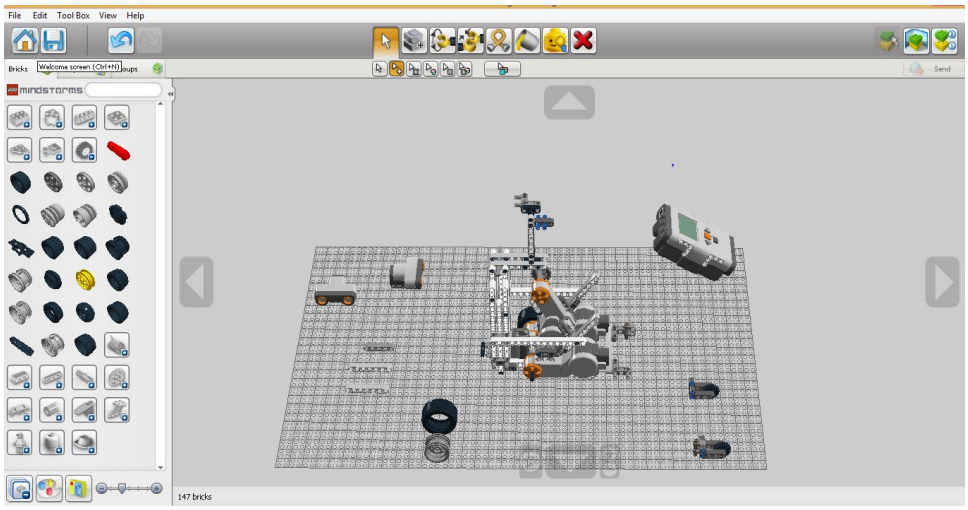


Figure 2.10: Lego Digital Designer

Upon completion of the model, the Lego Digital Designer was used to create a digital model of the robot. In addition, a documentation with descriptions about the robot was generated. The total number of parts is recorded one by one and it can take only minutes to re-create the model. The complete documentation is included in the DVD

provided with the project report. Figures D.1, D.2, D.3, D.4, **??**, 2.16 depict various phases of the designing process. It is worth mentioning that by the time the digital design started, it had been decided that the front Ultrasonic sensor should be placed facing backwards to avoid using the North wall since it was not necessary any more.

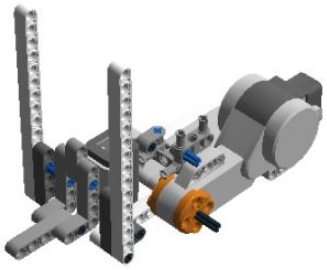


Figure 2.11:
LDD Stage A

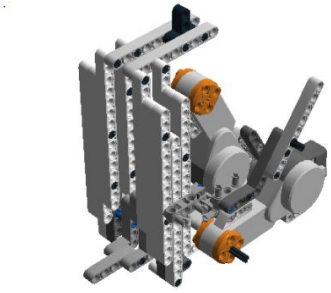


Figure 2.12:
LDD Stage B

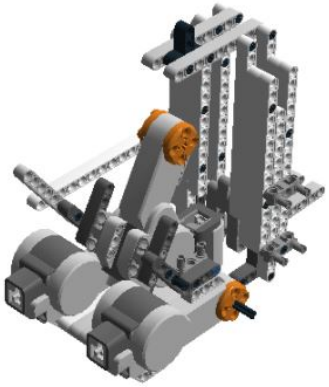


Figure 2.13:
LDD Stage C

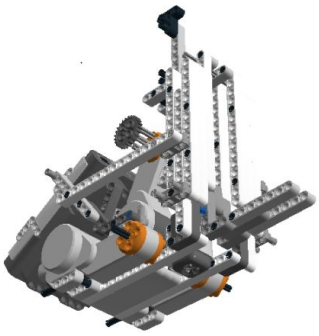


Figure 2.14:
LDD Stage D

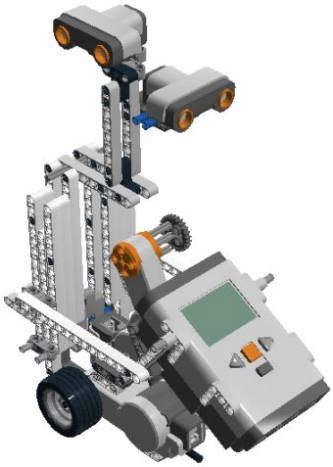


Figure 2.15:
LDD Stage E

Figure 2.16:
LDD Stage F

2.4 Conclusions

Right after the prototype designs the main project plan was created. The goals were now clear and the background research had already provided a lot of information to approach possible difficulties. The following chapters examine the three main objectives as conceived during the designing phase. Those are to design a GUI and program its functionality, to program the robot to perform the desired tasks and finally to achieve communication between those two.

Chapter 3

Implementation

The following chapter includes every step taken towards the implementation of the project. In more detail, it contains both the Graphical User Interface and the Robot code development. The development of the code run by the robot is divided into separate activities related to the various sensors to be used. Each sensor was tested thoroughly in order to ensure about its reliability.

3.1 Graphical User Interface

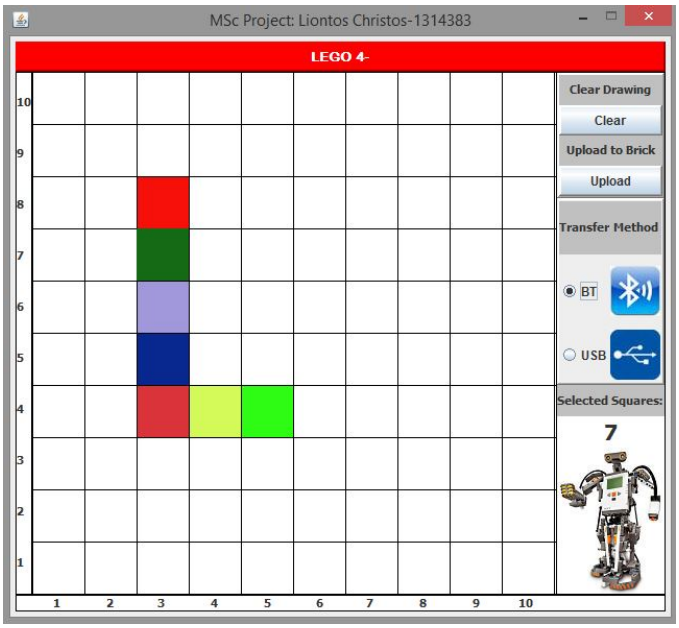


Figure 3.1: The Project Graphical User Interface

The Project's Graphical User Interface (Figure 3.1) is a simple Java GUI application designed to allow the user interact with the robot. It depicts a $10x10$ Grid that acts like a design editor. Its purpose is to inform the robot about the specific locations to be visited. It does so by transferring coordinates to the brick. The robot then uses these specific coordinates to place the bricks accordingly. The GUI is simple and offers very limited functionality. As mentioned before, its only purpose is to transfer the pairs of coordinates from the computer to the robot. Each square is selected individually. Every time a square is selected, it is filled with black color to differentiate itself from other non-selected squares. By the time a pattern has been drawn the user can upload it to the robot. This will initiate the building procedure. The data is transferred either via Bluetooth or USB technology. Below there is a brief description related to the GUI design and its specifications.

3.1.1 Software Used

During the project investigation, LeJOS NXJ was decided to be used for programming the robot. As mentioned before LeJOS is a Java Virtual Machine allowing the user to program using the Java Programming Language. As for the IDE, Eclipse was the best solution since it includes a LeJOS plugin and supports a very extensive API. Below there is a description of the required installations and configurations:

1. Install Java Development Kit (JDK) on the computer. JDK can be downloaded from
http://www.oracle.com/technetwork/java/javase/downloads/.
2. Download and install the suitable USB driver (Fantom driver) from the official LEGO website
http://www.lego.com/en-gb/mindstorms/downloads/software/
nxt-fantom-driver/.
3. Download and install LeJOS NXJ software from the official page
http://www.lejos.org/nxj-downloads.php.
4. Install Eclipse IDE. Eclipse can be downloaded from
http://www.eclipse.org/downloads/.
5. Install LeJOS plug-in to create NXT projects using Java.
6. Flash the NXT firmware to replace it with the LeJOS firmware.

3.1.2 GUI Development stages

Version 1. Drawing the Grid

The first version of the GUI (Figure 3.3) was a simple Grid drawn in the center of the frame [2]. The user was capable to select squares, however nothing would happen since the square's color would not change yet.

```
public void drawGrid(Graphics2D g2Buffer) {
    int w = this.getWidth();
    int h = this.getHeight();
    g2Buffer.setColor(Color.black); //xGrid, yGrid = 50
    for (int i = 0; i < w; i += xGrid) {g2Buffer.drawLine(i, 0, i, h);}
    for (int i = 0; i < h; i += yGrid) {g2Buffer.drawLine(0, i, w, i);}}
```

Figure 3.2: Drawing Grid on the canvas

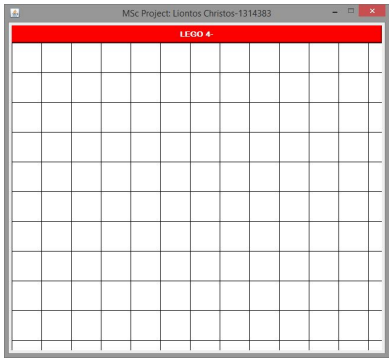


Figure 3.3: GUI Version 1

Version 2. User Selection of Cells

In version 2 (Figure 3.5), the selected squares would be painted black [2] in order to be differentiated by non selected ones. By the time the user clicks on a specific position inside the Grid, the x, y coordinates are sent to the draw method. A method then scales them and identifies the specific square inside which the user has clicked. Finally, the whole square is painted black. In addition, a buttons panel was added to the right part of the frame. A clear button was responsible for erasing any already drawn pattern on the frame. An upload button was also added, however without supporting any functionality yet.

```
private float scaletoXgrid(int x) {
    return (float) (((int) x) / xGrid) * xGrid;}

private float scaletoYgrid(int y) {
    return (float) (((int) y) / yGrid) * yGrid;}

public void setStartPoint(int x, int y) {
    this.xPressed = scaletoXgrid(x);
    this.yPressed = scaletoYgrid(y);}

public void draw(int x, int y,  Graphics2D g2) {
    setStartPoint(x, y);
    drawSquare((int) this.xPressed, (int) this.yPressed, g2);}

private void drawSquare(int x, int y, Graphics2D g2Buffer) {
    g2Buffer.fillRect(x, y, xGrid, yGrid);}
```

Figure 3.4: Painting the squares

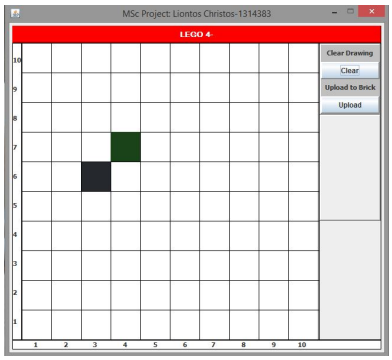


Figure 3.5: GUI Version 2

Version 3,4. Storing the Coordinates of Selected Squares

Versions 3 and 4 had the most significant updates as far as it concerns the functionality of the Interface. The main structure was developed to store the selected squares into arrays for future use. Figure 3.7 depicts the method which translates the x,y coordinates of the mouse and scales them to square-coordinates. Each time the mouse is pressed the exact x,y coordinates are saved in xPressed and yPressed variables. If x is between 0 and 50 then the corresponding square should be in the first column counting from the left (Grid x=1). If y is between 51 and 100 then the corresponding square should be in the second line counting from the upper left corner (Grid y=9).The intersection of these two means that the mouse was clicked somewhere inside the (1,1) square.

The GUI was now able to perform the required activities. The only thing had left was the implementation of the classes that would achieve a connection with the robot. In addition, versions 3, 4 included a few changes in the layout as well. Two rulers were

added to inform the user about each square's coordinates as well as a label to indicate the current number of selected squares.

```
public void setCoordinates(int xCoordinate, int yCoordinate) {
    int finalX = 0, finalY = 0;
    if (xCoordinate < 50) {finalX = 1;}
    else if (xCoordinate < 100) {finalX = 2;}
    else if (xCoordinate < 150) {finalX = 3;}
    else if (xCoordinate < 200) {finalX = 4;}
    else if (xCoordinate < 250) {finalX = 5;}
    else if (xCoordinate < 300) {finalX = 6;}
    else if (xCoordinate < 350) {finalX = 7;}
    else if (xCoordinate < 400) {finalX = 8;}
    else if (xCoordinate < 450) {finalX = 9;}
    else if (xCoordinate <= 500) {finalX = 10;}
    //Create a new point (xPressed, yPressed)
    point = new Point(finalX , finalY);
    //PointsArray checks for duplicate squares
    if (pointsArray.contains(point) == false) {
        pointsArray.add(point);
        xArray.add(finalX); yArray.add(finalY);}
```

Figure 3.6: Storing selected squares in Arrays

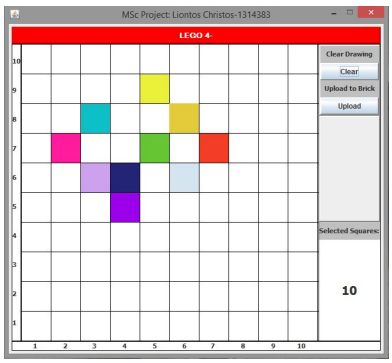


Figure 3.7: GUI Version 4

Version 5. Communications

In final version (Figure 3.9), the classes responsible for transferring data from the computer to the robot were added. The user is now capable of selecting a transfer method (BT or USB) and upload the data to the robot. In addition, additional code has been added to validate different steps while using the Interface. For example, what happens if one choose the same square twice, or what if upload button was pushed without selecting a transfer method.

```
public void send() {
    NXTConnector conn = new NXTConnector();
    conn.addLogListener(new NXTCommLogListener(){
        public void logEvent(String message) {
            System.out.println("BTSend Log.listener: "+message);}
        public void logEvent(Throwable throwable) {
            System.out.println("BTSend Log.listener - stack trace: ");
             throwable.printStackTrace();}});
    // Connect to any NXT over BlueTooth
    boolean connected = conn.connectTo("btspp://");
    if (!connected) {System.err.println("Failed to connect to any NXT by BT");
        System.exit(1);}
    //Create the input, output objects to be used to send the data
    DataOutputStream dos = new DataOutputStream(conn.getOutputStream());
    //Firstly we will dispatch the first element which is the number of the selected squares
    try {dos.writeInt(pointsArray.get(0));dos.flush();} catch (IOException ioe) {
        System.out.println("IO Exception writing bytes:");
        System.out.println(ioe.getMessage());}
    //We close the connection and create a second output object to re-initialize the procedure
    try {dos.close();} catch (IOException ioe) {
        System.out.println("IOException closing connection:");
        System.out.println(ioe.getMessage());}
    System.out.println("Number of points: " + pointsArray.get(0));
    //Create a second output object
    DataOutputStream dos2 = new DataOutputStream(conn.getOutputStream());
    //Now we send the elements starting from the second one
    //xArray elements and then yArray elements
    for(int i=1;i<pointsArray.size();i++) {
        try {dos2.writeInt(pointsArray.get(i));dos2.flush();} catch (IOException ioe) {
            System.out.println("IO Exception writing bytes:");
            System.out.println(ioe.getMessage());break;}}
    //Close the connections
    try {dos2.close();conn.close();}catch (IOException ioe) {
        System.out.println("IOException closing connection:");
        System.out.println(ioe.getMessage());}}
```

Figure 3.8: Transfering the information via Bluetooth

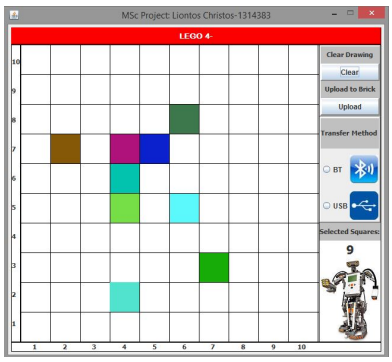


Figure 3.9: GUI Version 5

3.1.3 GUI Architecture

The GUI is consisted of five java classes. Those are the ProjectFrame.java, the DrawingPanel.java, the SquareDrawing.java, the BTSend.java and the USBSend.java. Figure 3.10 depicts the relationship between those classes. Below there is a brief introduction on the functionality of each one of them.

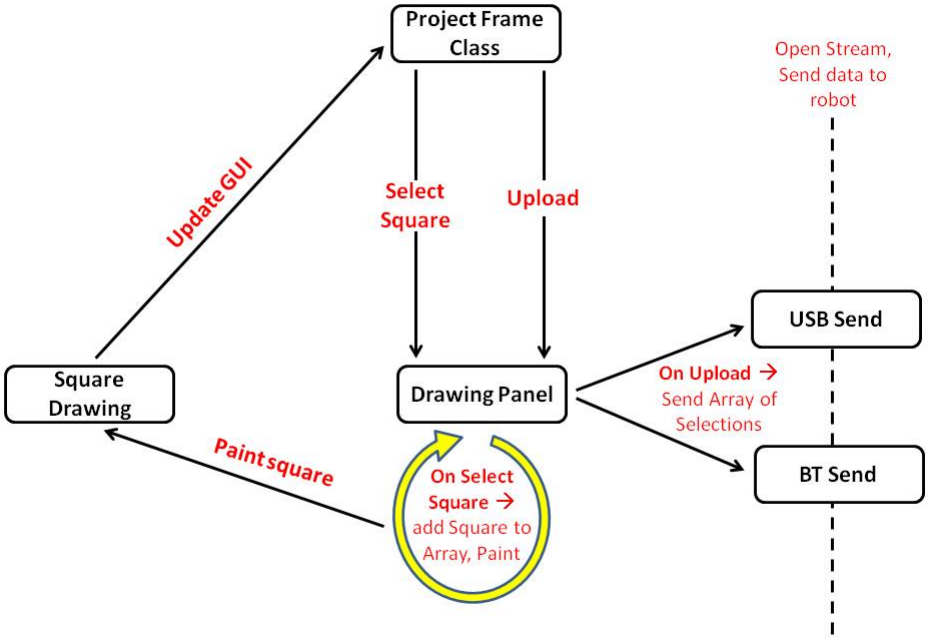


Figure 3.10: Block Diagram of GUI Application Classes

The Project Frame Class

The ProjectFrame.java class is a JFrame. It is the main frame that initiates the application and hosts every part in it. To avoid inconsistencies, and since it is a better practice, we do not add elements directly to the frame itself. A JPanel, named 'contentPanel', is being created to host the desirable functionality and it is this panel which is later added to the JFrame. The contentPanel has a Border layout. Thus, any panel added on, must be located to five main directions, East, West, South, North and Center. A Lego-label panel is located to the North, while two ruler-like panels to the East and South. The last two are used to create the impression of a Cartesian Coordinate System, allowing the user to read the coordinates of the each square. In addition, a buttonsPanel is located to the East hosting four buttons and one dynamically updated label. The first button is the Clear button which allows

the user erase any drawn pattern from the grid. The Upload button is responsible for initiating the communication between the computer and the robot. By the time it is selected, the coordinates of the selected squares are transferred to the robot. The following radio buttons, BT and USB, control the communication method the user wishes to use. Finally, a JLabel exists to depict the current number of selected squares. However, the most interesting part of the Frame is the hookingPanel, which is located at the Center of the contentPanel. It is an empty JPanel used as a canvas, on which drawing methods are applied. And since drawing on it requires to override the JPanel's PaintComponent methods, the only way is to create an individual class (DrawingPanel.java) and hook it up later to the hookingPanel (Figure 3.11).

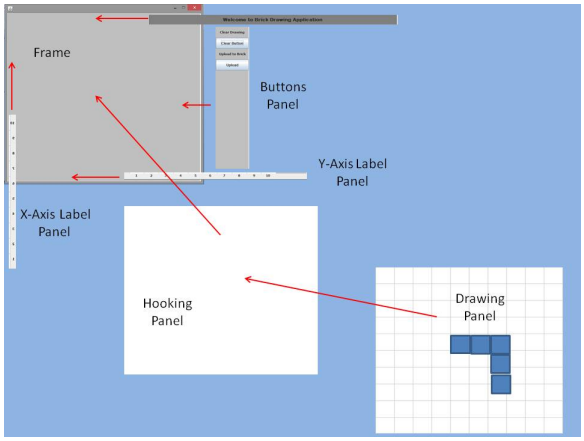


Figure 3.11: GUI Application Frame Components

Graphics Display and Interaction

The DrawingPanel.java is a JPanel. Whatever is depicted in the center of the application is drawn in this panel which is automatically hooked up to the hookingPanel of the JFrame class. The panel is used to draw the Grid on the canvas and deal with the events occurring when the user clicks on a specific location on the Grid. By the time the application starts, a drawGrid() method draws automatically a $10X10$ grid on the panel. The drawGrid method was described in version 1.

The panel then waits for a mouse event. When an event occurs (the user clicks somewhere on the grid) two methods are invoked. The setCoordinates() method takes as two parameters the coordinates of the specific position the mouse was clicked. It then translates these coordinates in square coordinates starting from one to ten. If the mouse coordinates are (0<=x<50, 0<=y<50) the selected square has to be the (1, 10), since each square has width = 50 pixels. Each time, the selected square values are saved in an Array for later use. The setCoordinates method was described in version 2.

The draw() method is invoked by a separate class named SquareDrawing.java. This class is responsible for painting black the selected square. This way, selected squares will be differentiated from non-selected ones. The draw method was described in version 3,4.

Painting onto the Canvas

The SquareDrawing.java is a separate class that contains the second method to be invoked when a mouse event occurs. This is the draw() method and is responsible for painting a square black, by the time the user clicks on a specific position inside its area.

Bluetooth and USB Communications

These classes are designed to deal with the communication between the GUI and the robot. The user may choose which class to be used by selecting one of the radio buttons. By the time the Upload button is clicked, an ArrayList containing the X and Y coordinates of the selected squares is sent the corresponding class. The send() method is then invoked that initializes a connection between the robot and the computer. If the robot has already started waiting for a connection, the ArrayList will be transferred to the brick using a DataOutputStream and a DataInputStream object. The two classes act exactly the same way with each other. The only difference is the transfer protocol the use. Below the is a sample code of the Bluetooth Send() method described before. This part of the method sends the first element to the robot (the number of selected squares). Right after, the remaining elements will be sent using a for-loop. The BTSend method is described in version 5.

3.2 Using the Sensors

Using the sensors is one of the most essential parts of the project. They are the only element that ensures the interaction between the robot and the environment. Thus, their reliability should be proven before proceeding to the actual implementation. As mentioned before in the design chapter, the robot's desired functionality related to the sensors is the following:

1. Follow lines,
2. Measure distances,
3. Move parallel to the East wall (Maintain a straight route).

In order to achieve the above abilities, the procedure is broken into sort activities that allow for the sensors to be tested separately. The following subsections examine the decisions taken during the implementation phase.

3.2.1 Using the Compass Sensor

Moving inside the frame requires a reference point. A reference point is a spot that remains constant during the process and lets the robot compare its current position towards it. No matter how many changes have been applied to the robot's position, it will always be able to find its current direction by searching the reference point and calculating the difference between the two spots. There are plenty of types of reference points to be used such as objects, magnetic fields etc. Finding a constant element will always depend on the desired project and sensors available.

The first experiment will be to use the compass sensor for navigating inside the frame. Thus, North will be used as a reference point. The robot will be able to keep track of its current position by measuring the difference in degrees from North. Assuming that the compass works accurately, North will always remain in the same direction at 0 degrees. Lejos-API provides a CompassHTSensor class. It is a class that allows the robot to control the compass and read its values. Figure 3.12 indicates the sensor values in various positions.

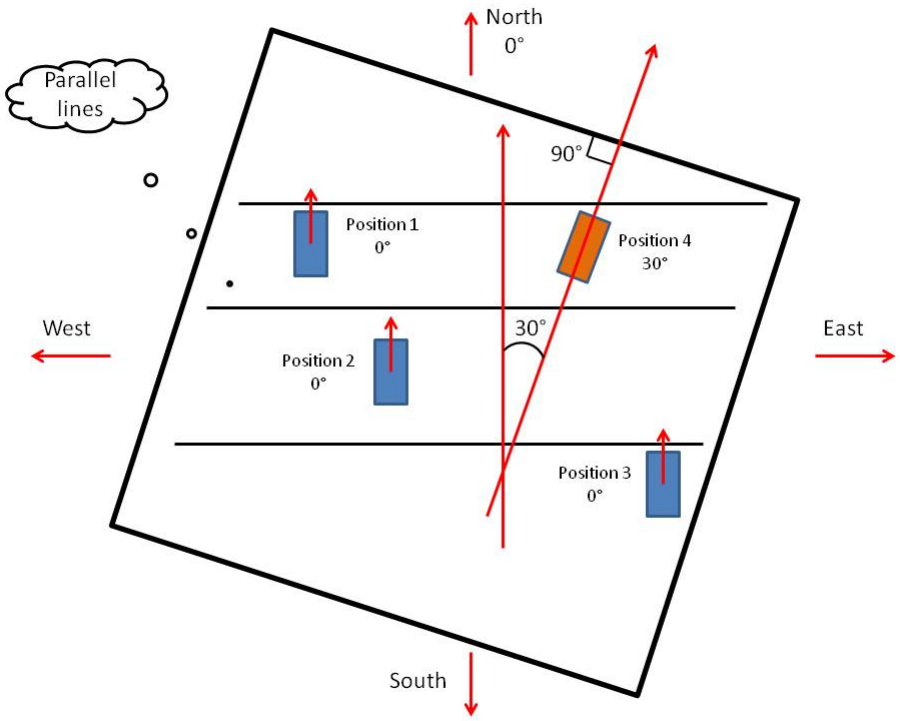


Figure 3.12: Sensor values in various positions

While in position one, two or three, the sensor indicates 0 degrees from North. This is actually because the robot is facing to the North. Even if the position changes, the values should remain the same as long as the robot moves in parallel lines. As you will notice, while in position four, the sensor indicates 30 degrees from North. This means the direction of the robot changed by rotating approximately 30 degrees clockwise. Having a better look at the picture, one can realize that while in position four, the robot is facing towards the upper wall. And that is the power of the reference point. By locating North and 0 degrees the robot is now able to find the desired location by applying a specific degree rotation. Thus, if we need it to find the upper wall, a command rotate to 30 degrees from North will be enough to place it towards the wall. A very useful function provided by the API is the resetCartesianZero(). This method changes the current direction the compass is facing to the zero angle. Thus, the perception the robot has about North is different now, as the compass indicates 0 degrees when facing to the initial direction. This way, one can place the North-reference point wherever it suits better. In the example above, it would be more suitable if the robot would reset the Cartesian zero while facing towards a wall. That would create a Cartesian coordinate system where each wall would be at 0, 90, 180, 270 degrees respectively.

Tracking and maintaining a direction

The robot can perform a straight movement quite accurately without decline from its route. However, rotating for specific number of degrees is not always that easy. The need to rotate inside the frame is extremely significant, thus a solution should be found in order for the robot to be able to perform 90 degree turns accurately. Unfortunately turning depends on many factors which are not always easy to control. Lego connections are not very strong, thus components always have to tested to ensure they are stuck tight together. Wheels for example should be safely secured on the motors, because a slight change in the track-width (the distance between the centers of the two wheels) may lead to a larger rotation. In addition, if the robot bumps into a very small obstacle it can also make it decline of its route by a small amount of degrees. However, even if it is a small deviation, it is still a problem since after two meters of straight movement this small difference will grow much bigger. For example let us say that a robot is facing to the East. The compass indicates the value of 270 degrees. It performs a 90 degree right turn and is supposed to face now to the North. However, there is a very small difference and instead of 0 the compass indicates 1. The robot initiates a forward movement but two meters away it is clear that it is not moving accurately towards North since the compass now indicates 3 degrees (Figure 3.13).

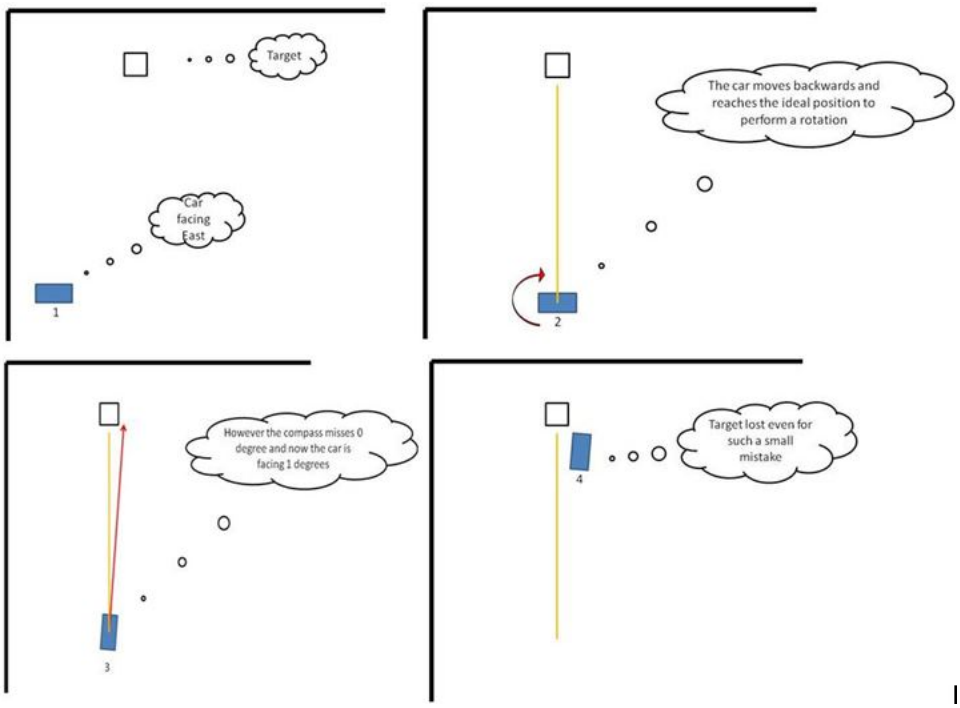


Figure 3.13: Approaching a target using the compass sensor

In order to solve such problems an Auto Correct Route algorithm should be developed in association with the compass. Using the compass, the robot can track a specific direction. By that point, it is essential to maintain this direction until it reaches a desired spot. This will ensure that the movement is as straight as possible. The picture above indicates a behavior in four steps. The objective is for the robot to reach the square close to the North wall. Initially, it faces towards the West wall. It then moves backward for a specific distance (the X-coordinate of the square). It rotates 90 degrees until it faces the North wall. Finally, it travels forward until the square is reached (the Y-coordinate of the square). As mentioned before, if no compass is used, the robot may rotate less, or more degrees than it is desired. In addition, even if it happens to turn accurately, there is no guarantee that it will continue moving in a straight line. Fortunately, the Lejos API provides a CompassPilot class that controls the motors in cooperation with the compass. Using the getCompassHeading(); command the robot can track its current direction and rotate by 90 degrees left or right. Then, a travel(distance)(); or forward(); command can ensure that it will travel for a specific distance (or forever) while the initial direction is maintained by steering left or right. Algorithm 2 was designed to demonstrate the power of the compass pilot class. It performs the above behavior. However, it is worth mentioning that it was developed as an initial experiment to achieve satisfactory movement.

Algorithm 2: Moving from dispatch point to square position using compass

while *distance from the East wall* < 50 **do**
    Move Backward;
end
Stop;
Rotate 90 degrees (using the compass);
while *distance from the North wall* > 60 **do**
    Move Forward using the compass forward(); method to maintain the direction;
end
Stop;

```java
public class CompassPilotTest {

    private float trackwidth = 120.1f;
    private float wheeldiameter = 49.6f;
    private CompassHTSensor compass;
    @SuppressWarnings("deprecation")
    private CompassPilot pilot;
    private UltrasonicSensor s;

    @SuppressWarnings("deprecation")
    public CompassPilotTest(){
        compass = new CompassHTSensor(SensorPort.S1);
        pilot = new CompassPilot(compass , wheeldiameter , trackwidth , Motor.A , Motor.C , false);
        s = new UltrasonicSensor(SensorPort.S2);
    }
    @SuppressWarnings("deprecation")
    public void travel() {

        float startingDegrees;
        float desiredDegrees;
        int distance;
        //Get current direction
        startingDegrees = pilot.getCompassHeading();
        //Get current distance
        distance = s.getDistance();
        //Travel backward until desired distance is reached
        while (distance < 50){
            pilot.backward();
            distance = s.getDistance();
        }
        pilot.stop();
        //Calculate desired degrees (90 degrees clockwise)
        desiredDegrees = startingDegrees + 90;
        //Control the values
        if (desiredDegrees >= 360){
            desiredDegrees -= 360;
        }else if (desiredDegrees < 0){
            desiredDegrees += 360;
        }
        //Rotate to desired degrees
        pilot.rotate(desiredDegrees);
        //Create a new zero to maintain it
        pilot.resetCartesianZero();
        //Get current distance
        distance = s.getDistance();
        //Travel forward until desired position is reached
        //while maintaining the direction
        while (distance > 60){
            pilot.forward();
            distance = s.getDistance();
        }
        pilot.stop();
```

Figure 3.14: Sample code that implements Algorithm 2

TroubleShooting

During the first experiments, trying to get the robot reach the (10, 10) square using the above techniques, the following problem was discovered. The initial model had the Ultrasonic sensor right beneath the compass. After some tests from various positions were held, it was discovered that the Ultrasonic sensor could not detect the correct distance when over 170 cm. The result was having the same value (255 which is the default value when the sensor is not operating anymore) for any distance greater than 170 cm approximately. The code did not seem to have any kind of logical faults at that time. It was discovered that the compass somehow created a magnetic field or something else that prevented the ultrasonic sensor from operating in bigger distances. By accidentally changing the position of the Ultrasonic sensor and placing it further

from the compass seemed to have solved the problem (Figure 3.15). However, it is still not clear what caused the sensor's malfunction and no similar situations have been found online.

Figure 3.15: Compass and Ultrasonic sensor picconet

Testing the Compass sensor's reliability

Using the above technique for navigation requires the compass to function properly giving steady values no matter how many times the program is running. A series of tests were taken to prove its reliability before proceeding to further implementation of the project. The sensor was positioned several cm away of the motors and the brick to avoid magnetic interference (Figure 3.16). Also, before each test the compass had been calibrated to reduce the possibility of getting inaccurate results.

Figure 3.16: Design used for testing the sensor's reliability

Testing 90° rotation accuracy without resetting Cartesian zero.

During the following test, the robot performed a number of sort movements to investigate the accuracy of rotations. It used the compass sensor to measure a 90° rotation starting from a random position towards North. In other words the reference point remained the actual North. After each rotation the difference between the initial and final angle, as well as the corresponding error were measured. The results are depicted in the Table 3.1.

Algorithm 3: 90° rotation without resetting Cartesian zero

Get current angle;
Rotate 90 degrees using the compass to measure; Stop;
Get current angle;
Get error of rotation;

```
public void travel() {
    float startingDegrees;
    float firstStop;
    float desired;
    pilot.setTravelSpeed(40);
    pilot.setRotateSpeed(30);
    startingDegrees = pilot.getCompassHeading();
    System.out.println( startingDegrees);
    desired = startingDegrees + 90;
    if (desired >= 360){desired -= 360;
    }else if (desired < 0){desired += 360;}
    //rotate method from CompassPilot
    pilot.rotate(desired);
    firstStop = pilot.getCompassHeading();
    System.out.println("Ending " +  firstStop );
    error = pilot.getHeadingError();
    System.out.println("error " + -error);
    pilot.rotate(error);}
```

Figure 3.17: Compass test without resetting the Cartesian zero.

Starting Angle	**Ending Angle**	**Error**	**Did it look like 90° rotation?**
21	111	1?	NO
18	106	2	YES
17	105	2	NO
-50	39	2?	NO
-2	90	1	NO
-145	-53	2	YES
-158	-68	0	YES
-63	-26	1	NO
15	103	2	NO
108	-161	1?	NO
15	103	1?	NO
48	138	0	YES
11	102	1	NO
11	100	1	NO
11	102	1?	NO
11	103	2	NO
12	103	1	NO
-150	60	0	YES
-57	32	1	NO
-18	72	0	NO

Table 3.1: Compass test results without resetting Cartesian zero.

Testing 90° rotation accuracy while resetting Cartesian zero.

The second series of tests were conducted under the same conditions. The only difference was that the Cartesian zero was now set to the initial angle. In other words, the robot had the perception that in its initial position it faced North. This is obvious from the corresponding Table 3.2 since the initial value is always equal to zero.

Algorithm 4: 90° rotation resetting Cartesian zero

Reset Cartesian zero;
Get current angle;
Rotate 90 degrees using the compass to measure;
Stop;
Get current angle;
Get error of rotation;

```
public void travel() {

    float firstStop;

    pilot.resetCartesianZero();
    pilot.rotate(90);
    firstStop = pilot.getCompassHeading();
    System.out.println("Ending " +  firstStop );
    this.error = pilot.getHeadingError();
    System.out.println("error " + error);
    pilot.rotate(error);
}
```

Figure 3.18: Compass test resetting the Cartesian zero

Starting Angle	**Ending Angle**	**Error**	**Did it look like 90° rotation?**
0	91	1	NO
0	91	1	YES
0	89	1	NO
0	90	0	NO
0	89	1	NO
0	91	1	NO
0	88	2	NO
0	92	3?	NO
0	91	1	YES
0	92	1?	NO

Table 3.2: Compass test results resetting Cartesian zero.

Tables 3.1,3.2 above indicate the values the sensor got during the tests with and without resetting the Cartesian zero. One can notice that sometimes the calculations seemed to be incorrect. But even when they were correct, according to the sensor, the actual angle did not look like 90 degrees at all. No accurate measurements had to be taken since only by visual observation it was clear that the deviation was higher than 20 degrees. Further tests would have been conducted if such difference would not have been obvious visually. It is worth mentioning that after any rotation an additional rotation was applied to correct the error. However, the results were still disappointing since the possibility of performing an accurate 90 degree rotation were below 25%.

Testing accuracy in maintaining direction.

The following test was conducted to investigate the accuracy of a straight movement using the compass sensor. The robot should read its initial angle towards North and maintain it while moving forward for a specific distance. An already existed method would take care of the process by steering left or right whenever a deviation had been detected.

Algorithm 5: Maintaining the direction

Set current angle to the one to maintain;
Travel for the specified distance maintaining this angle;
Stop;
Get current angle;

```
public void travel(){
    pilot.setTravelSpeed(40);
    pilot.setRotateSpeed(30);
    System.out.println("Starting" + pilot.getCompassHeading());
    pilot.setHeading(pilot.getCompassHeading());
    pilot.travel(1000);
    System.out.println("Ending" + pilot.getCompassHeading());}
```

Figure 3.19: Maintaining the direction

Starting Angle	**Ending Angle**	**Was it straight?**
108	88	NO
17	25	NO
65	82	CLOSE
93	93	YES
100	158	NO
43	7	NO
-54	-57	CLOSE
-78	-76	CLOSE
-54	-53	YES
-57	-45	NO

Table 3.3: Compass test results while forward movement is performed.

Table 3.3 indicates the values in degrees before and after the movement was applied. A ruler was used to observe if the movement was straight or not. One can observe that the possibility of performing a straight movement using the current compass is almost 50%. This makes it a very unreliable option for the project.

Test the compass in parallel positions.

During the fourth test, two rulers (approximately 3 meters length) were used as straights lines. The rulers were placed parallel to each other. According to the theory, placing the compass (with the same angle each time) on any point on the rulers should give the same values. However, the results were still disappointing since the deviation was quite big. A very common fact was that if for example the compass indicated 23 degrees on a spot A while 29 on a spot B, getting back to the A spot after some seconds would indicate a 30 degree, even though it was the exact same spot tested before.

Conclusions

A second electronic compass was used as well to figure out whether the problem inside the house is generic. The results were almost the same since the compass seemed to have great variations and especially when close to walls. Sometimes, the values were jumping even 70 degrees without any change in the direction. The problem may be due to the various magnetic fields that may interfere with the compass inside the house (WI-FI, Steel objects like tables and chairs, cables inside the walls, etc). Similar issues have been found in on other projects [3]. Test No4 was held in some University rooms as well but no differences have been observed. There were only a few similar situations found online [4], thus the most possible explanation is that the compass is malfunctioning only because of the close environment conditions. A mechanical failure could not be proved. However, in any case the current compass seems to be unreliable for the specific project. Even if a method could be found to reduce the possibilities of malfunction, it will still be risky to use it. Alternative solutions should be found to make navigation more precise and malfunctions less often.

3.2.2 Using the Ultrasonic Sensor

Since the compass proved to be unreliable for navigating inside the frame, an alternative solution is necessary to achieve the task. As mentioned before, a reference point is needed for the navigation. Thus, a sensor is needed, that will be capable of recognizing this reference point. Since the magnetic field-reference point is no more an option, the robot is not able to determine its position. Having a look at the perimeter walls, the most logical reference point left to be used would a wall itself. The Ultrasonic sensor has already been used before to measure the distance from the North wall. However, using the Ultrasonic sensor means that the robot's movements should be much more limited that before. The robot should always be aware of which wall it is facing. And

since there are four walls this may become much more complicated if it is allowed to move freely inside the perimeter. An easy solution would be to design the robot to be capable of facing only to the West and North wall. That means that the other two walls will not impact the robot's navigation strategy. Algorithm 6 describes the same behavior introduced before, but without the use of a compass sensor.

Algorithm 6: Moving from dispatch point to square position using the tachometer, compass

while *distance from the East wall* < 50 **do**
    Move Backward;
end
Stop;
Rotate 90 degrees (using the tachometer);
while *distance from the North wall* > 60 **do**
    Move Forward using the compass forward(); method to maintain the direction;
end
Stop;

Maintaining a direction

It is obvious that the tachometer is now necessary to achieve the 90 degree rotation. However, as mentioned before, using the tachometer may end up rotating the robot for less or more degrees than the ones desired. In advance, this will lead to moving towards a false direction, deviating from its straight route. As a conclusion, an Auto Correct Route algorithm should be developed for a second time to make the robot capable of maintaining its direction. The problem can be solved by adding a second Ultrasonic sensor in a vertical position. This means that the first sensor will be aiming in front of the robot, while the second one to the side of it (Figure 3.20). By continuously tracking the distance of the West wall, the robot will be capable of maintaining an almost straight route. However, there is not a specific method that can perform this action, so the algorithm should be developed from scratch. Algorithm 7 describes a possible successful version as far as the sensors function properly, without returning inaccurate values.

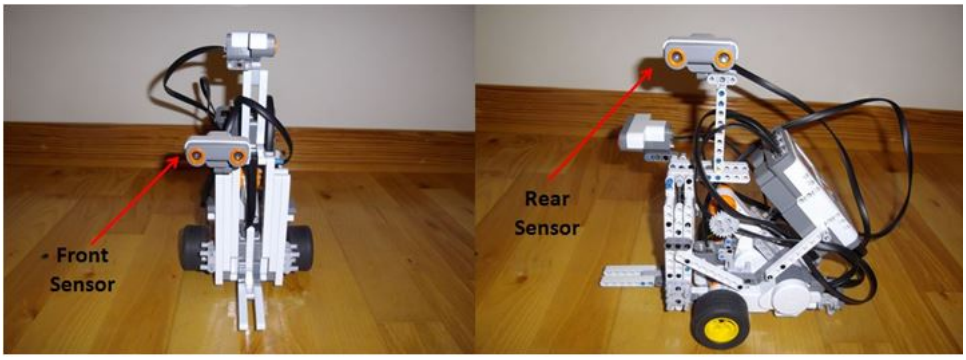


Figure 3.20: Ultrasonic sensor formation

Algorithm 7: Moving from dispatch point to square position using the tachometer, Ultrasonic

while *distance from the East wall* < 50 **do**
    Move Backward;
end
Stop;
Rotate 90 degrees (using the tachometer);
while *distance from the North wall* > 60 **do**
    if *distance from the East wall* $= 50$ **then**
        Move forward
    else if *distance from the East wall* < 50 **then**
        Steer slowly to the right
    else if *distance from the East wall* > 50 **then**
        Steer slowly to the left
end
Stop;

```
// Go forward, keeping a specific distance from rear wall. If the
// distance changes make modifications until you maintain it again
temporaryRearDist = RearSonicSensor.getDistance();
temporaryBackDist = BackSonicSensor.getDistance();
while (temporaryBackDist < desiredDistFromFront_YCoordinate) {
    // If you go straight, means that you maintain the distance
    while (temporaryRearDist == distToMaintain
            && temporaryBackDist < desiredDistFromFront_YCoordinate) {
        pilot.forward();
        temporaryRearDist = RearSonicSensor.getDistance();
        temporaryBackDist = BackSonicSensor.getDistance();
        if (temporaryBackDist == 255) { temporaryBackDist = 200;}}
    // If your direction changes and distance is lost it means
    // you no longer go straight. Thus steer left or right until
    // you find this distance again
    while (temporaryRearDist != distToMaintain
            && temporaryBackDist < desiredDistFromFront_YCoordinate) {
        if (temporaryRearDist > distToMaintain) { pilot.steer(10);
        } else if (temporaryRearDist < distToMaintain) { pilot.steer(-10);
        } else if (temporaryBackDist >= desiredDistFromFront_YCoordinate) {
            break;  }
        temporaryRearDist = RearSonicSensor.getDistance();
        temporaryBackDist = BackSonicSensor.getDistance();
        if (temporaryBackDist == 255) { temporaryBackDist = 200;}}
    temporaryRearDist = RearSonicSensor.getDistance();
    temporaryBackDist = BackSonicSensor.getDistance();
}
pilot.stop();
```

Figure 3.21: Maintaining direction sample code

The most interesting part of the code is the steering procedure. As far as the distance between the sensor and the wall is steady, the robot moves forward. However, it is almost impossible that it will keep moving parallel to the West wall. By the time the robot deviates from its straight route there are two possibilities. It will either steer slightly to the left or to the right. Steering to the left means that the distance between the sensor and the wall is now smaller than it was before. Thus, the sensor values should inform the robot that it should now steer slightly to the opposite direction until the distance becomes the same again. On the other hand, if the robot happens to deviate from its route by steering to the right, the distance will become greater than the desired one. The same way, the robot should steer to the left to correct its route. Although the algorithm is not the most sophisticated that could be used, it works quite accurately as long as the sensor is not malfunctioning. One of its advantages is that even if the robot does not perform an exactly 90 degree rotation, the algorithm will later correct the difference since it will have to steer left or right. On the other hand, a main disadvantage is that when it is positioned far away from the wall, the values tend to be more inaccurate since the angle the sound is reflected from the wall is bigger. Figure 3.22 indicates how the robot performs.

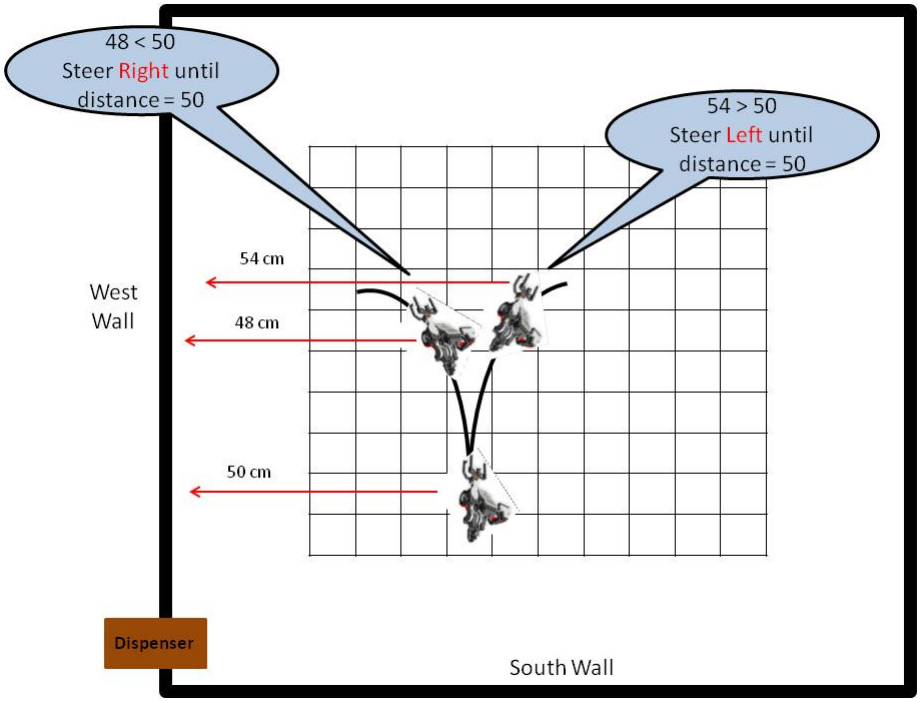


Figure 3.22: Maintaining the direction

Testing the Ultrasonic sensor's reliability

The Ultrasonic Sensor is one of the most essential parts of the robot. It is responsible for reading various distances between the robot and the Frame-walls. Thus, it is the main navigation system used by the robot to determine its exact position while functioning inside the frame. It is obvious that failure in defining the correct distance in centimeters, or even a slightly small deviation, could lead to a total failure of the project, since a brick could be placed inside the wrong square. For this reason, several tests should be conducted to ensure the proper functionality of the sensor. It is worth mentioning that the sensor's capability of reading the correct distance is also depended to the material it faces. Too shiny or too rough surfaces both tend to provide wrong data feedback. For this project, the material to be used for the frame-walls is Styrofoam (Aerogel Thermal Insulation Material) due to the fact that it is very light and can be cut into pieces very easily.

The testing was conducted using the Ultrasonic sensor and a tape (Figure 3.23). The sensor runs a program that gets the distance from the frame wall and displays it in the NXT Brick screen. It continues reading and writing until the user terminates the procedure. After several measurements, it was obvious that the sensor is functioning

properly giving most of the times quite accurate results. However, the following events were spotted:

1. After running the program for a while, the sensor seems to malfunction occasionally. Although it mostly reads the distances accurately, sometimes it may give completely inaccurate values. This may be due to the fact that from time to time the sensor may needs re-calibration. In addition, when batteries are fully charge the errors seem to occur more rarely. Thus, it would be wise always to run the program with as much power as possible, for more accurate results.

2. While moving, the sensor keeps receiving information. This is a live functionality so it invokes the time parameter as well. This means that there may be a little delay between the moment the robot should act and the moment it actually acts. For example, by the time the sensor is being moved from the position of 60 cm to 30 cm, there is a delay displaying the correct distance on the screen. There are many possible solutions to this problem such as reducing the speed of the robot in order to borrow it some more time to process the sensor's values. A second alternative would be the use of threads which increases the speed of data processing. This is because threads allow multiple tasks to run at the same time. Thus, reading distances and moving will be two separate activities running in parallel. Finally, an algorithm should be developed to ensure correction movements in case the robot does overshoots the desired position.

3. When the distance becomes greater than 170 centimeters the values seem to introduce a small deviation. The difference is approximately one cm and it does not seem to change while the distance becomes even bigger, which is quite strange. For example, when the real distance is 175 cm the monitor depicts 176 and when it is 200 the monitor depicts 201 cm. The difference is still one cm, which means it does not change while the distance changes. Although this fact does not introduce any serious problems, it is still a malfunction one should take under consideration if extremely accurate results are needed.

4. There are occasions where the Ultrasonic sensor fails to read any distance. In such situations it will return a default value 255. This means that the distance is either greater than 255, or the sensor fails to detect an object. The designed frame is $2X2$ meters. Thus, the distance will never be greater than 255 cm. As a result, whenever the 255 value is returned, it is a sign of malfunction. The sensor will not stop operating though. It will keep searching until it detects an object. This usually

takes less than a second. However, it is enough to confuse the robot and get it perform something incorrectly.

5. The Styrofoam material offers many advantages for this project. It is light and can be cut easily, making it the best solution, since the frame will have to be moved for the presentation. However, it is not the most sound-reflective material to be used [5]. After running some simple tests, moving the sensor in various positions in front of the foam, it was observed that it is not reliable at all. The various angled surfaces seem to absorb a good deal of the emitted sound waves. Thus, the sensor failed to read the correct distance. Assuming that the sensor is not malfunctioning any more the material was dressed with tin-foil, a thick layer of Aluminum. Tin-foil is a thick layer of aluminum that offers great heat insulation. It is not the most sound-reflective material though. However, it is a cheap solution that would transform the foam surface to a smoother, harder one. Hard and smooth surfaces tend to reflect sound more efficiently [6]. The results were quite satisfying and the errors almost disappeared.

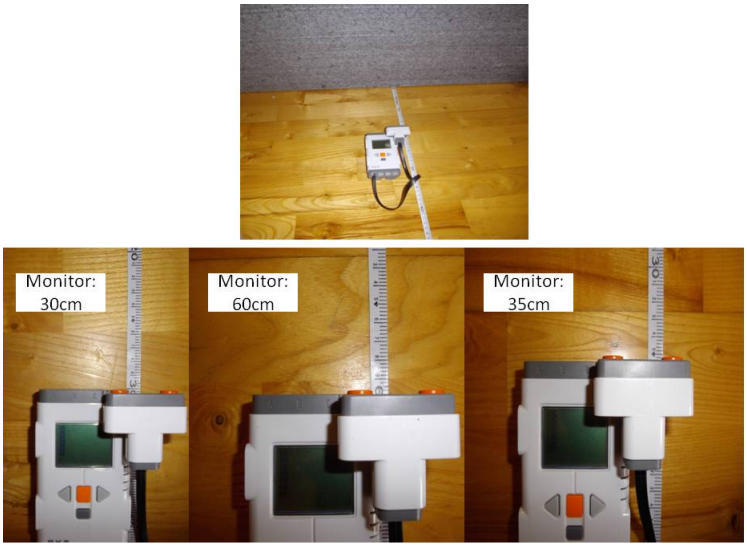


Figure 3.23: Ultrasonic sensor tests

Conclusions

Combining the above problems all together may lead to the complete failure in the procedure. Each one had to be solved separately to reduce the possibilities of malfunctions at most. During the implementation of the project, the following actions were performed to achieve the desired results:

- The Robot was used fully charged most of the times to ensure that any flaws were not a result of power failure. (No 1)
- Threads were used to access the values as fast as possible. In addition a correction algorithm was added in case the robot overshoots the desired distance. (No 2)
- A filter was applied to prevent getting the default 255 value. An if statement controls the values by rejecting the specific 255. (No 4)
- Assuming that the sensor is not malfunctioning any more, the wall material was dressed with tin foil. The idea was that tin foil is a fine insulation material offering a shiny surface. It is mostly used for heat insulation. However, it was obvious that it would reflect sound much more effectively than foam. The results were quite satisfying as the errors were almost disappeared.

3.2.3 Using the Color Sensor

Moving towards the brick dispenser has to be a very accurate movement. The robot should be always capable of placing the fork-lift under the current brick. However, since the brick's width is not very big, even a slight deviation on its route will have as a result an empty lift. Thus, a solution should be found to secure this action. A good option is to use a color sensor to track a line on the floor. If the line leads to the desired location then the problem is solved. Following a line can be achieved in various ways.

The best option would be using a black tape (the line) and two color sensors with a small distance between them (Figure 3.24). The distance should be equal to the tape's width in order for the tape to 'fit' inside the gap. Following the line, means that the two sensors should always get a different color than the tape's one (black). While moving, if the right sensor senses black, it means that the robot should steer left, while if the left sensor senses black it means it should steer right. In other words, the two sensors allow the robot feel the direction to which it is going to fail and thus turn towards the opposite one to correct the route. This is very secure way to follow a line, even if there are sharp turnings.

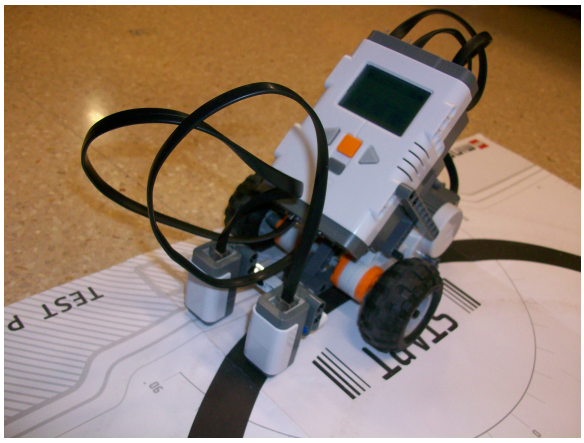


Figure 3.24: Line follower using two color sensors [7]

However, for this project, there is only one color sensor available to be used. Thus, an alternative solution should be found. Using one color sensor makes the task even more complicated (Figure 3.25). The robot should perform the same way as before, with the difference that now it is much more difficult to recognize the direction it needs to steer since it gets data from only one sensor. As a result, it cannot differentiate right from left. Most line following algorithms would use the zig-zag method according to which the robot moves in a z-like movement to allow the sensor scan the tape horizontally. While moving, if it encounters a different color after performing a left rotation, it knows that it faces to the left side of the tape. Thus, it has to rotate right. However, rotating right means that after some time it will reach the right side of the tape. But since the last rotation was to the right, it will now rotate left. This is a sequence of continuous changes in direction, in an effort to scan the line .

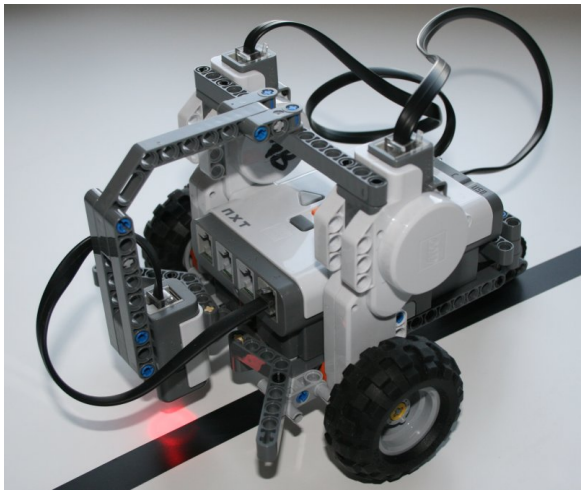


Figure 3.25: Line follower using one color sensor [8]

Since for the specific project there is no need for sharp turns to be performed, a different approach will be attempted. The line needed to be followed is a straight one. Three different color tapes will be placed on the floor one next to each other (Figure 3.26). The robot will be programmed to follow the yellow one which will be the central tape. By the time the sensor senses a different color, a slight steering to the left or right will be applied, according to the input color. As mentioned before, there is no need for sharp movements, so steering by a small amount of degrees will place the robot again on the yellow line. A small correction code may be added to adjust the route even more accurately. By the time the robot enters the yellow line while coming from the black one, it should steer to the opposite direction immediately instead of waiting to reach the red line. This way, it will be capable to follow a straight route accurately and place itself under the brick. Below, Algorithm 8 describes the line following procedure while Figure 3.27 depicts the performance.

Figure 3.26: Line following solution for the project

Algorithm 8: Following the line

while *current distance* $\neq$ *required distance* **do**
    if *Yellow line* **then**
        Move forward
    end
    if *Red line* **then**
        Steer left. When yellow line is found, steer a bit right.
    end
    if *Black line* **then**
        Steer right. When yellow line is found, steer a bit left.
    end
end
Stop;

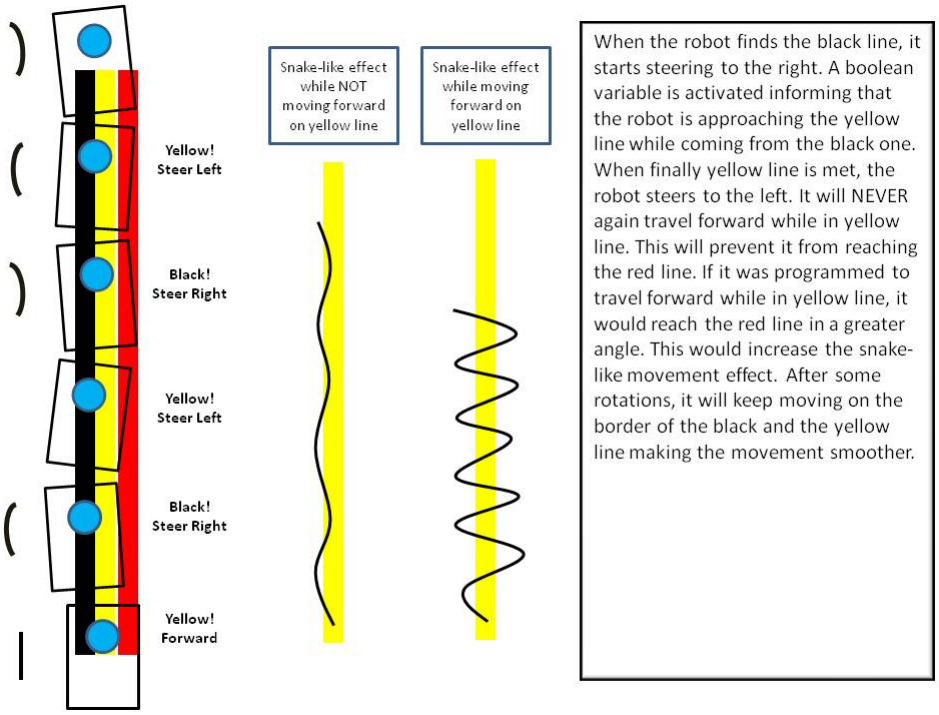


Figure 3.27: Line following explained

Testing the Color sensor's reliability

Before proceeding to the implementation of the project, the color sensor was thoroughly tested to check its reliability. More than five tapes of different colors were placed on the floor. The brick run a sample program which could indicate the exact color pointed in front of the sensor. In addition, it depicted the RGB values, which means we could distinguish between similar variations of the same color (dark blue, light blue etc.). The aftermath of the tests was that the sensor is functioning properly returning accurate values for any color. However, the following problems were spotted.

1. When it came to test the color of different surfaces rather than clean, shinny tapes, the results were not that satisfying. For example, when pointing to the floor the return color was red despite the fact that the actual wood color was light brown. It is worth remembering though that the color sensor will be used to recognize the tape's colors. Thus, even if it cannot function properly in more complex surfaces, it would not introduce any problems during the implementation.

2. Additionally, the sensor sometimes fails to read a color while the robot is moving. This may lead to missing a location where it was supposed to stop. According

to the project, the robot will be programmed to freeze near the dispatch point. This will be achieved by using a blue tape at the desired position. Failure to stop will have as a result overshooting the desired position making it impossible correct the mistake. The sensor was re-programmed using threads, to avoid missing values while the code runs in a loop. However, it did not seem to solve the problem which still occurs occasionally.

Conclusions

As a conclusion, the color sensor is capable of reading the tape's colors quite accurately. However, while the process is running, occasionally it may skip some values. If the desired color to read is not related with a sensitive action, then it will not be much of a problem. For example, while following the line, even if one value is missed, the robot will continue with the next one since it keeps moving. On the other hand, since there is a limited time-window to search for the blue color, failing to read its value immediately will have as a result missing the dispatch point were it was supposed to freeze.

3.2.4 Using Threads

During the project investigation a short research was conducted as far as it concerns the Threading techniques. Threads are widely used in programming especially when robots are involved. They offer the flexibility to handle more than one event at the same time. Every program includes a series of actions happening one after the other. However, there are times when the program is needed to perform actions in parallel. This is where threads are used. Even though our robot does not perform a too much complicated action, threads are still useful.

The most complicated actions performed by the robot occur when it needs to use two or more sensors at the same time. Let us examine the case were the robot needs to follow the line until it reaches the desired x-coordinate. During this process there are three main events happening:

1. Using the motors to move.
2. Using the Back Ultrasonic sensor to examine if the desired distance is reached.
3. Using the color sensor to follow the line.

During the implementation of the program the first option was to program the robot without the use of threads. Nested while-loops were used to control the sensors

and allow them functioning continuously when needed. As a result, at least two inputs should always be examined inside one loop. The first one was from the Back Ultrasonic sensor while the second one from the Color sensor. The code was quite complicated but that was not the main problem. After running a series of tests it was clear that the robot mostly overshot the x-coordinate and failed to read the colors on time. In other words the robot moves so fast that the brick does not manage to process the inputs from the sensors on time.

As slowing down the speed of the robot was not an option, an alternative should be found using threads. The solution came when every sensor was programmed separately in a different thread. As a result, three threads do now control the one color and two ultrasonic sensors. The functionality of each thread is a simple read-value method that continuously updates a global variable. Figure 3.28 depicts the relationship between the Classes and Threads of the program.

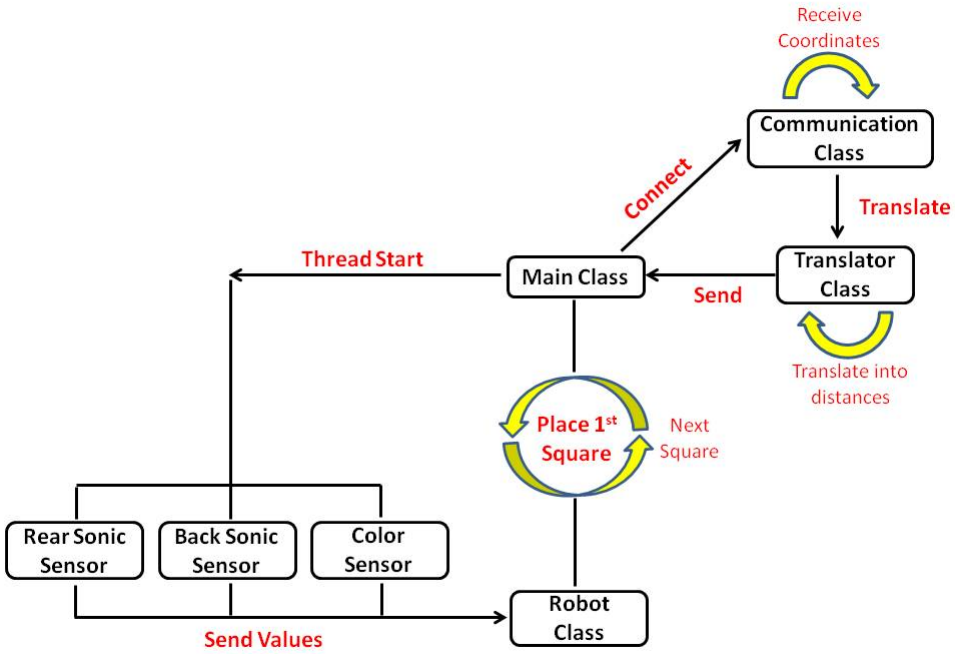


Figure 3.28: Block Diagram of Robot Application

Conclusions

As a conclusion, the following advantages were observed:

- The complexity of the code has been reduced,
- Sensors operate through separate programs without interfering with each other,
- The speed and accuracy of performance has been increased

3.3 The flow of Execution

The above sections examined the main functionality of the robot and the possible ways to achieve it. The Ultrasonic and color sensors were tested and proved reliable to be used. Thus, the robot is capable of following lines, measuring distances and moving parallel to walls (maintain direction). The only thing left is to develop a code that will bind all those short activities together in a logical succession.

Below follows a brief presentation on how the data flow from the computer to the robot and how the second one translates them into specific actions. This section is critical in order to understand the whole procedure. This involves both GUI and robot events as well as the flow of data between them. However, only the most sensitive parts of the procedure will be described, to avoid getting confused by insignificant details. As mentioned before, the robot is programmed to execute a standard building procedure in cooperation with the Graphical User Interface. Despite the fact that both the robot and the computer run two different programs, it is essential to understand that the first one cannot operate without the second. This is because the robot needs specific instructions before starting the operation. Figure 3.29 depicts the activities diagram that describes the sequence of the on-going activities.

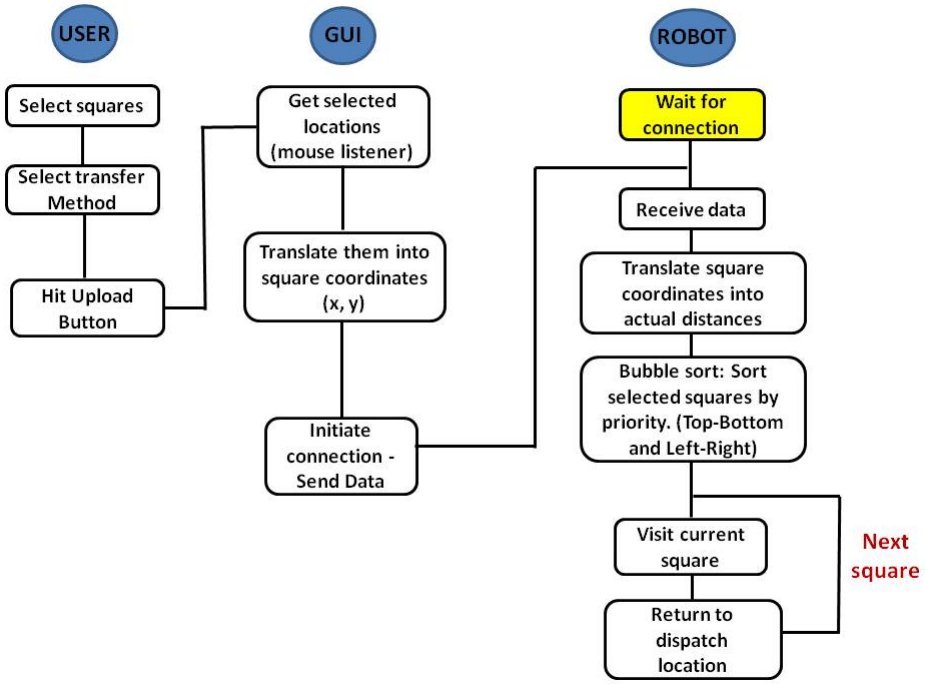


Figure 3.29: Activities Diagram

3.3.1 The GUI

As mentioned before, the GUI offers very limited functionality. This is because its only job is to let the user communicate with the robot and give it specific directions according to where exactly he wants the bricks to be placed. It depicts a 10x10 grid which is actually a replica of the real grid that is drawn on the floor. By the time the user clicks on a square on the GUI, the x, y coordinates of the mouse are saved on a temporary variable. However, these values have no meaning yet. Thus, they are translated immediately in a second pair of coordinates that will define the specific square. To make it more clear, let us say that the user selects the(1, 1) square (clicks somewhere inside its area). Since the program is not yet aware of that, the variable saves the mouse coordinates which are x=35 and y=460. Taking under consideration that Java starts counting the pixels from the upper left corner of the application and each square has a 50 pixel width, a simple method can calculate that the mouse should have been clicked somewhere inside the (1, 1) square. Concluding, these two values are stored in a Java ArrayList. This is the Arraylist to be send later to the robot when the upload button will be selected.

3.3.2 The Data transfer

The data transfer is achieved using input and output streams. The robot is not programmed to communicate back to the GUI. Thus, only the computer can send data to the robot. By the time the brick is turned on and the building program is selected to run, the robot opens an input steam and waits for a possible connection either via USB or Bluetooth. The user has the ability to choose the desired connection via a text-menu that appears on the LCD. Each time the upload button in the GUI is clicked, an output stream is initialized to send the data to the robot which is already waiting. It is worth mentioning that if using BT connection the two devices should be paired, otherwise NXT will not be visible to the GUI. The first value to be sent is the number of selected squares. By the time this is stored in a variable, a second output will be used to send the exact squares selected. As mentioned before, those values are stored inside an ArrayList.

3.3.3 The Robot: Translation phase

The robot now holds some information about the desired squares. However, a number of paired values do not have any special meaning for it. The sensors operate using real distance values measured in centimeters. So, it is obvious that the coordinates should be translated again into different pairs. A translator method is responsible for performing this function. The coordinates ArrayList passes as a parameter to that method which transforms the values into actual distances. That means that the developer is already aware of these distances and has already measured extra spaces from the sides of the grid (50 cm). For example, the (1, 1) pair of coordinates will be translated approximately to (50, 90) which will depict the real distance between the square and the East, South walls. As an aftermath, the robot is now aware that in order to locate this specific square, it needs to move in a position where the distance towards the East wall will be close to 50 cm while the one towards the South wall 90 cm. Every single pair is translated and stored inside another ArrayList<Points> which will be used later.

3.3.4 Arranging the order of brick placement

It is worth mentioning, an interesting event that occurs before the translation. The user is most likely to select the squares in a random way inside the grid. The ArrayList that stores the initial coordinates will only be aware of the order the user chose these squares. However, it would be a serious problem to program the robot locating these squares in the same order the user chose to. This could lead to a much more

complicated code without any reason. For example, let us say that the user chooses firstly the (5, 5) square and secondly the (5, 8) one. What would happen if the robot had to place a brick on the (5, 5) square at the first time? It would block the road for the (5, 8) square making it really complicated to place the next brick. The answer to this problem is to sort the ArrayList in such a way that the robot will never face difficulties in moving freely towards the next square. A reasonable idea is the bricks to be placed with a priority, from top to bottom and left to right. Translating this into an algorithm means that the Points should be sorted by descending order as far as it concerns the y-coordinate and by ascending order as far as it concerns the x-coordinate. As a conclusion, the square with the highest y-value and the lowest x-value will be placed first, following with the next one. The code below indicates how the Bubble sort method was used to sort the ArrayList. The way it works is checking two values each time and swapping them if the first one is bigger (or smaller, depends on the action we need to perform). The same process occurs until no more swaps are necessary. Figure 3.30,3.31 depict how it actually works.

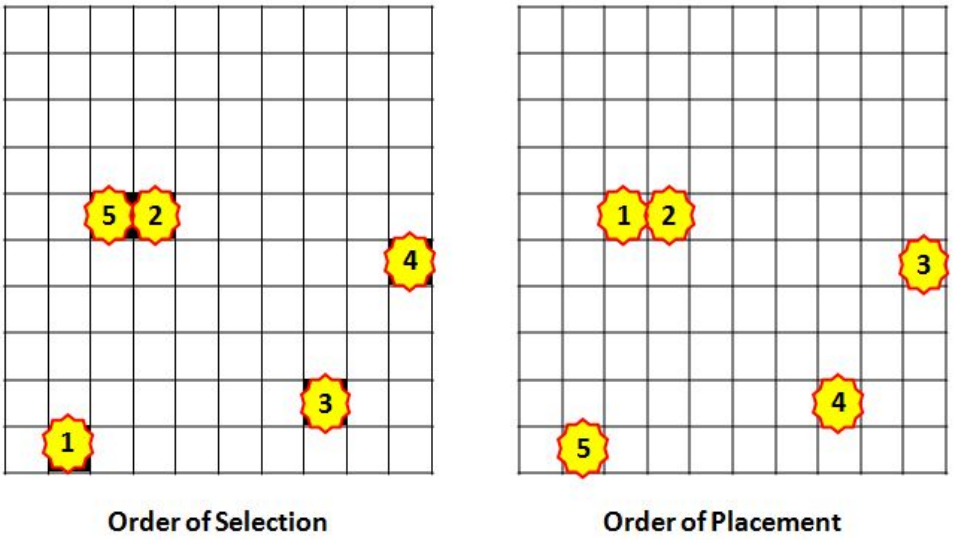


Figure 3.30: Selection and Placement order

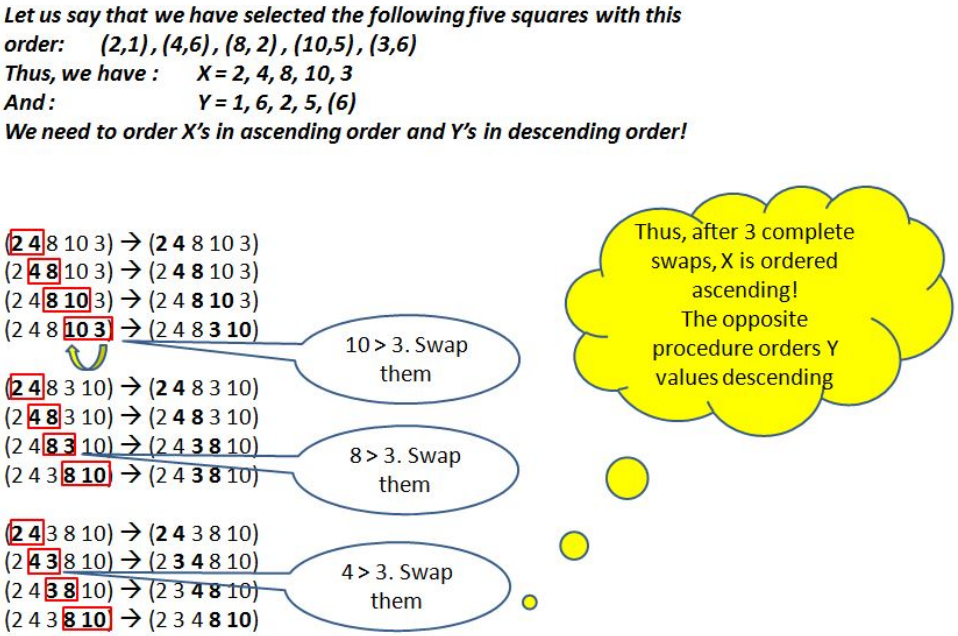


Figure 3.31: Ordering X's in ascending order

```
public void sortPoints() {
    Point bigger = null;
    Point smaller = null;
    boolean sign = true;

    while (sign) {
        sign = false;

        for (int i = 0; i < pointsArray.size() - 1; i++) {
            //Sort by bigger Y (From top to bottom)
            if (pointsArray.get(i).getY() < pointsArray.get(i + 1).getY()) {
                smaller = pointsArray.get(i);
                bigger = pointsArray.get(i + 1);
                pointsArray.set(i, bigger);
                pointsArray.set(i + 1, smaller);
                sign = true;
            //If the points have the same Y, sort by smaller X (from right to left)
            } else if (pointsArray.get(i).getY() == pointsArray.get(i + 1).getY()) {
                if ((pointsArray.get(i).getX() > pointsArray.get(i + 1).getX())) {
                    bigger = pointsArray.get(i);
                    smaller = pointsArray.get(i + 1);
                    pointsArray.set(i, smaller);
                    pointsArray.set(i + 1, bigger);
                    sign = true;
                }
            }
        }
    }
```

Figure 3.32: Implementation of BubbleSort

3.3.5 The Robot: Building phase

The robot now has all the desired information required to initiate the building procedure. It is aware of the exact locations it needs to visit and the specific order it has to follow. Thus, it is now time for the sensors to enter the game. The robot's initial position is in front of the brick dispenser. A loop begins updating each time the square that is about to be visited. The algorithm that exists inside the loop is divided in two actions, the PlaceBrick() method and the ReturnToDispatchPoint() method.

Brick Placement

This part of the code will take care of the current brick placement. By the time the procedure has begun, the fork-lift starts elevating the brick off the floor. The robot then performs a 180 degree rotation until it faces the opposite direction. It then starts moving forward as far as it satisfies two conditions. The first one is that the distance from the East wall is not yet equal to the desired one (x-coordinate) while the second one is that the robot is following the line. When the desired distance is reached, it then performs a 90 degree rotation until it faces the North wall. This is when the route-maintaining algorithm will be invoked. The robot starts moving forward until the desired distance is reached (y-coordinate). While moving, the distance from the East wall should remain steady, thus moving in a straight route. Finally, it stops in front of the selected square and lowers the fork-lift in order for the brick to be placed on the floor.

```
// Travel method. Starts from the dispatch point. Moves the robot back
// until a desired distance is reached (X-coordinate of square). Turns 90
// degrees. Moves forward until a desired distance is reached (Y-coordinate)
public void placeBrick(int x, int y) {
    this.desiredDistFromRear_XCoordinate = x; //Parameters come through main class
    this.desiredDistFromFront_YCoordinate = y;
    this.distToMaintain = (desiredDistFromRear_XCoordinate + 7);
    int temporaryRearDist = 0;
    int temporaryBackDist = 0;
    int colorID;
    boolean comingFromBlack = true;
    boolean comingFromRed = false;
    pilot.setTravelSpeed(travelSpeed);
    //Lift the fork
    Motor.B.rotate(300);
    //Rotate 180 degrees
    pilot.rotate(2*nintyDegrees);
    //While on the line, move until desired distance is reached
    temporaryBackDist = BackSonicSensor.getDistance();
    while (temporaryBackDist < desiredDistFromRear_XCoordinate) {
        colorID = ColourSensor.getID();
        while (colorID == red){{
            if (comingFromBlack){pilot.steer(-40);comingFromBlack = false;
            }else if (comingFromRed){pilot.steer(40);comingFromRed = false; }
            temporaryBackDist = BackSonicSensor.getDistance();
            if (temporaryBackDist == 255) {temporaryBackDist = 200;
            }else if (temporaryBackDist == desiredDistFromRear_XCoordinate){ break;}
            colorID = ColourSensor.getID();}
        colorID = ColourSensor.getID();
        while (colorID != red){
            if (colorID == black) { pilot.steer(40);comingFromBlack = true;
            } else if (colorID == white) {  pilot.steer(-40);comingFromRed = true; }
            temporaryBackDist = BackSonicSensor.getDistance();
            if (temporaryBackDist >= desiredDistFromRear_XCoordinate) {break;
            } else if (temporaryBackDist == 255) {temporaryBackDist = 200;}
            colorID = ColourSensor.getID();
            temporaryBackDist = BackSonicSensor.getDistance();}
        temporaryBackDist = BackSonicSensor.getDistance();}
```

Figure 3.33: Brick placement-Part 1

```
pilot.stop();
//Rotate 90 Degrees to face the grid
pilot.rotate(nintyDegrees);
// Go forward, keeping a specific distance from rear wall. If the
// distance changes make modifications until you maintain it again
temporaryRearDist = RearSonicSensor.getDistance();
temporaryBackDist = BackSonicSensor.getDistance();
while (temporaryBackDist < desiredDistFromFront_YCoordinate) {
    // If you go straight, means that you maintain the distance
    while (temporaryRearDist == distToMaintain
            && temporaryBackDist < desiredDistFromFront_YCoordinate) {
        pilot.forward();
        temporaryRearDist = RearSonicSensor.getDistance();
        temporaryBackDist = BackSonicSensor.getDistance();
        if (temporaryBackDist == 255) {temporaryBackDist = 200;}}
    // If your direction changes and distance is lost it means
    // you no longer go straight. Thus steer left or right until
    // you find this distance again
    while (temporaryRearDist != distToMaintain
            && temporaryBackDist < desiredDistFromFront_YCoordinate) {
        if (temporaryRearDist > distToMaintain) {pilot.steer(10);
        } else if (temporaryRearDist < distToMaintain) {pilot.steer(-10);
        } else if (temporaryBackDist >= desiredDistFromFront_YCoordinate) {break;}
        temporaryRearDist = RearSonicSensor.getDistance();
        temporaryBackDist = BackSonicSensor.getDistance();
        if (temporaryBackDist == 255) { temporaryBackDist = 200;}}
    temporaryRearDist = RearSonicSensor.getDistance();
    temporaryBackDist = BackSonicSensor.getDistance();}
pilot.stop();
// If you overshoot the required distance travel back until you fix it
temporaryBackDist = BackSonicSensor.getDistance();
if (temporaryBackDist > desiredDistFromFront_YCoordinate) {
    pilot.setTravelSpeed(approachingSpeed);
    while (temporaryBackDist > desiredDistFromFront_YCoordinate) {pilot.backward();
        temporaryBackDist = BackSonicSensor.getDistance();}
    pilot.stop();}
Motor.B.rotate(-300);}
```

Figure 3.34: Brick placement-Part 2

Returning to Dispatch Point

The last part, performs more or less the reverse actions of the above method. The robot has already placed the brick to the desired position. It starts moving backward as far as it satisfies two conditions. It should firstly maintain the straight route using the same algorithm as previously. The only difference is that now it moves backwards. In addition, it should keep moving backwards until the color sensor finds the line. By the time the line is found, it stops immediately and performs a 90 degree rotation to the left. It should now face the East wall. The final part of the method is to follow the line until the robot reaches the dispenser. A blue or green electrical tape is used to inform the sensor that the robot has reached the desired location. When the robot finds this tape, it stops immediately having the fork-lift under the next brick to be placed. The line should be followed strictly in order for the fork-lift to be placed under the brick. Slight rotations to the left or right may end up missing the correct spot. As

long as the robot has reached the dispenser, the next loop will start, performing the same two methods for the next brick.

```
public void returnToDispatchPoint() {
    int colorID, temporaryRearDist = 0;
    boolean comingFromBlack = true, comingFromRed = false;
    pilot.setTravelSpeed(travelSpeed);
    colorID = ColourSensor.getID();
    temporaryRearDist = RearSonicSensor.getDistance();
    while (colorID != red) {
        // If you go straight, means that you maintain the distance
        while (temporaryRearDist == distToMaintain && colorID != red) {
            pilot.backward();colorID = ColourSensor.getID();
            temporaryRearDist = RearSonicSensor.getDistance();}
        // If your direction changes and distance is lost it means
        // you no longer go straight. Thus steer left or right until
        // you find this distance again
        colorID = ColourSensor.getID();
        while (temporaryRearDist != distToMaintain && colorID != red) {
            if (temporaryRearDist > distToMaintain) {pilot.steerBackward(10);
            } else if (temporaryRearDist < distToMaintain) {pilot.steerBackward(-10);
            } else if (colorID == red) {break;}
            colorID = ColourSensor.getID();
            temporaryRearDist = RearSonicSensor.getDistance();}
        colorID = ColourSensor.getID();
        temporaryRearDist = RearSonicSensor.getDistance();} pilot.stop();
    pilot.rotate(nintyDegrees);
    colorID = ColourSensor.ID;
    while (colorID != blue) {
        colorID = ColourSensor.getID();
        while (colorID == red){
            if (comingFromBlack){pilot.steer(40);comingFromBlack = false;
            }else if (comingFromRed){pilot.steer(-40);comingFromRed = false;}
            colorID = ColourSensor.getID();}
        colorID = ColourSensor.getID();
        while (colorID != red){
            if (colorID == black) {pilot.steer(-40);comingFromBlack = true;
            } else if (colorID == white) {pilot.steer(40);comingFromRed = true;}
            if (colorID == blue){break;}
            colorID = ColourSensor.getID();}
        colorID = ColourSensor.getID();} pilot.stop();
```

Figure 3.35: Return to dispatch position

3.4 Conclusions

The implementation phase made it possible to identify difficulties and weaknesses and proceed to alternative solutions. Testing the sensors individually made it easier to decide which ones would suit best for the project. As an aftermath, the Compass sensor was replaced by a second Ultrasonic sensor to provide more accurate navigation. The final version of the robot application is a result of a combination between various activities that were programmed at the first stages of the implementation process. The following chapter includes the testing and evaluation of the completed version according to the objectives described in the project specification.

Chapter 4

Evaluation & Testing

The following chapter describes the testing of the final version of the project and the evaluation of its behavior according to the assigned objectives in the project specification. As mentioned before, the project was divided into two main parts, the Graphical User Interface and the Robot programming. Each one of them had to reach certain criteria to be considered successful. Those are mentioned below.

The GUI:

- Has to be simple to use
- Has to provide a Grid panel to the user allowing him to draw specific patterns
- Able to upload building instructions to the robot either by BT or USB connection

The Robot:

- Has to be able to operate inside the frame using Ultrasonic and Compass sensors
- Has to be able to receive instruction from the GUI either by USB or BT connection
- Has to be able to lift specially designed Lego bricks
- Has to be able to place bricks precisely according to the given instructions

Considering the above objectives several experiments had to be conducted in order to test and evaluate the functionality of the last version developed. The interface was the first part of the project to be designed since it was essential in order to continue working with the robot. It is a java application able to connect to the NXT brick and transfer specific data to it. Its functionality is fixed and tested several times during

the experimentation with the robot, thus no further tests had to be conducted. The following section describes the test conducted to investigate and evaluate the robot's behavior as developed in the final version of the application.

4.1 Testing the Robot's functionality

The final version of the robot program had to be tested thoroughly in order to decide if it is functional according to the assigned objectives. The robot was programmed in parts as already mentioned in the implementation chapter. Every individual activity had already been tested before the final version. During the implementation phase it was confirmed via several tests that the robot was able to perform the following activities successfully:

- Receive instructions by the GUI,
- Lift bricks,
- Find specific locations inside the frame and
- Move parallel to walls.

However, combining each of those activities into one that would contain all the above functionality required further testing. In order to do so, a number of predefined tests had to be run. Each test was decided to include the placement of five bricks on the grid. This way each experiment would last longer (approximately 5 minutes) in order to provide a continuous time, long enough to observe the robot's behavior. A number of activities were examined on every single placement within each experiment. There were times where the robot failed before reaching an end (Color sensor failed to recognize the blue color on dispatch point and overshot it). However, despite the fact that failure in one activity probably means total failure, the robot was placed again manually in the desired position in order to continue testing the remaining activities. Table 4.1 depicts the activities tested and the results given.

Forward parallel movement towards square	**Back Ultrasonic Sensor Malfunction**	**Backward parallel movement towards red line**	**Stopped on red-blue line**	**Alignment with the tape**	**Bricks placed correctly**	**Stopped earlier than dispatch**
5-5	0-10	4-5	2-2	SLOW	4	2-5
5-5	0-10	3-5	2-2	SLOW	5	1-5
3-5	0-10	4-5	2-2	SLOW	5	1-5
5-5	0-10	4-5	1-2	SLOW	5	1-5
5-5	0-10	3-5	2-2	SLOW	5	0-5
4-5	0-10	5-5	2-2	FAST	5	0-5
5-5	0-10	5-5	2-2	FAST	5	0-5
3-5	0-10	2-5	2-2	SLOW	5	0-5
5-5	0-10	4-5	2-2	SLOW	5	0-5
5-5	1-10	5-5	2-2	FAST	4	0-5
5-5	0-10	5-5	1-2	FAST	5	0-5
5-5	0-10	5-5	2-2	FAST	5	0-5
4-5	0-10	4-5	1-2	SLOW	5	0-5
5-5	0-10	2-5	2-2	SLOW	5	2-5
3-5	0-10	5-5	1-2	FAST	5	0-5
5-5	1-10	5-5	2-2	FAST	4	0-5
5-5	0-10	2-5	2-2	SLOW	5	1-5
4-5	0-10	5-5	2-2	FAST	5	0-5
5-5	0-10	1-5	2-2	FAST	5	0-5
5-5	1-10	5-5	2-2	FAST	4	0-5

Table 4.1: Final version test results.

4.1.1 Description of the Test-table

The above table records the most sensitive activities regarding the success of the procedure. In case each one of them does not get performed properly it may lead to a total failure of the procedure. Below is a brief explanation of each activity and the conclusions made through the testing phase. Each attribute of the table is examined and explained separately.

Forward parallel movement towards square

This activity involves the first parallel movement towards the current square to be visited. A perfect move is dependent on two basic factors. The first one is the line following activity. If the robot follows the line strictly, the 90 degree rotation applied

later will place it parallel to the West Wall. The second one is the Ultrasonic sensor. If the sensor performs properly, the robot will be able to steer left and right in time, correcting its route before it is too late.

Table value "YES" means that the movement was accurate and the robot stopped right in front of the square to be visited. On the other hand, value "Almost" means that the robot reached the target in an angle. Thus, the brick was not placed exactly inside the square. However, this may be considered as a flaw rather than a failure. Having a look at the results, such a flaw occurred 9 out of 100 times which is not a very small percentage.

Back Ultrasonic Sensor Malfunction

This activity examines the proper function of the Back Ultrasonic Sensor. The Back sensor is used twice for every square to be visited. It firstly searches for the x-coordinate, while following the line and then for the y-coordinate while approaching the desired square. The sensor thus is responsible for finding the exact location needed for the brick to be placed. Within each test, the sensor is used 10 times.

Examining the results, it is obvious that the sensor functions properly with a very small possibility of failure. Only 3 out of 200 times the robot failed to stop at the desired location. It is worth mentioning, that even if such an event occurs, the result will be a brick in a wrong position, which will not affect the upcoming activities. The procedure will go on for the next bricks.

Backward parallel movement towards red line

This is probably the most critical activity in the whole procedure. It involves the movement that begins after a brick has been placed and before the red line is encountered. The algorithm that controls the movement is almost identical to the one that controls the forward parallel movement towards the square. However, the mechanical design of the robot does not allow the same performance. Having a look at the shape of the robot, it is obvious that the motors are placed closer to the front part. That leaves the biggest part behind the wheels. When the robot moves forward, it can steer its whole body faster within a smaller angle. On the other hand, when moving backwards, the angle of rotation grows a bit bigger, creating the effect of a less smooth movement (snake effect is greater). Thus, if the sensor delays to return a value, the robot will not move strictly, parallel to the wall since it will always have to correct a bigger mistake in direction.

The activity is very important because it can generate further problems to the procedure. If the snake effect is great enough, the robot will reach the red line in an angle far away from 90 degrees. Thus, applying the 90 degree rotation which comes next, will not position the robot upon the line. As a result, it will take it more time to align with the tape creating a second snake effect while now following the line. This is quite obvious when looking at a next activity in the table, the speed of alignment. Whenever the movement is not accurate, the alignment with the tape is slow. A huge problem would appear, if the last square visited would be close to the dispatch point (x=1, x=2). The robot would delay aligning with the line and since there is not much distance until the dispenser, it would miss the brick. The results show that such events occurred 19 out of 100 times. This is close to a 20 percent possibility of failure. Small adjustments were made to reduce this possibility, however without great results.

Stopped on red-blue line

The following two attributes involve the same activity, though in different stages of the procedure. The color sensor is responsible for recognizing specific colors on the floor in order to inform the robot change its behavior. Following a line is a continuous effort of tracking the same color. Even if one input value is missed, a second one will appear immediately without affecting the performance of the movement. However, when the sensor is searching for a specific color which will appear only once for a very short time, it is extremely important to obtain this value. Failure to distinguish the color in time would result to a total failure since the robot will either bump to the wall or lose the dispatch location.

The table results show an up to 10 percent possibility of color sensor failure. Four out of forty times the sensor got used did not manage to stop at the right location. And considering the fact that no correction movements can be applied, there is a 10 percent possibility for the procedure to fail due to the color sensor alone.

Alignment with the tape

This attribute has already been discussed above. It describes the time the robot needs to align with the tape. As mentioned before, it is highly related to the back parallel movement towards the red line. It was observed that the speed was slow only when the robot failed to approach the red line in a vertical position. Thus, it is mostly a matter of a weak algorithmic approach to the problem, rather than a sensor malfunction.

Bricks placed correctly

The following column records the number of bricks placed correctly within each test. By terms correctly, it is meant that more than half of the brick is placed within the borders of the desired square. It is quite unlikely that the robot will approach the square in such accurate angle for the brick to be located in the center. As a failure will be considered the event during which a brick will be completely placed upon a different square. Looking at the results, that was most likely to happen when the back Ultrasonic Sensor failed to read the correct distance. However, the possibility for such events seems to be less than 4 percent which means it is something quite rare.

Stopped earlier than dispatch

The following attribute examines the ability of the robot to stop in the right position in front of the dispatch point. This column should not be confused with the "Stopped on red-blue line" one. It describes the possibility for the robot to stop before reading a value, rather than missing a value. In other words, there were occasions where the color sensor read the blue value while the blue tape had not been met yet. The problem may have occurred because black and blue are colors with their RGB values close enough. Thus, sometimes the sensor may confuse them. The tests show an up to 8 percent possibility of this event to occur, which can be decreased by changing the blue tape into a color with much more distant RGB values.

4.2 Project Evaluation

Taking under consideration the above findings, a number of conclusions were conducted that allowed to evaluate the complete form of the project. The evaluation is divided in two parts. The first one examines the Graphical User Interface functionality while the second one the Robot application.

4.2.1 Graphical User Interface Evaluation

The GUI was designed to be simple and easy to use. It depicts only a few buttons allowing the user to concentrate on the main functionality, which is the Grid. The user is capable to design a specific pattern by selecting individual squares. Selected squares are painted with a variety of random colors making them easily distinguished by non-selected ones. By the time the pattern is designed, it can be uploaded to the robot using

the Upload button provided. Before a pattern is uploaded, a transfer method (either BT or USB) should be selected, otherwise a pop up window will appear informing to do so.

Using the GUI is straight forward. Create a design and upload it to the robot. A usability test was conducted to enhance the justifications. Fifteen users were asked to use the Interface without any further instructions. However, they had already visual contact with the Frame and were informed they were dealing with a robot application. Table 4.2 records the results of the experiment.

Gender	**Age**	**IT Background**	**Questions Asked**	**Topic**	**Success**
Male	26	YES	0	-	YES
Male	24	YES	0	-	YES
Male	26	YES	0	-	YES
Male	26	NO	0	-	YES
Female	28	NO	1	Grid	YES
Male	22	YES	0	-	YES
Female	47	NO	2	Combo boxes	NO
Female	35	NO	1	Grid	YES
Male	54	NO	0	-	YES
Male	19	NO	0	-	YES
Male	23	YES	0	-	YES
Female	25	NO	0	-	YES
Male	27	NO	0	-	YES
Female	25	NO	1	Grid	YES
Male	17	NO	0	-	YES

Table 4.2: GUI Usability Test.

4.2.2 Robot Performance Evaluation

Concluding from the tests described above, the robot is able to perform most of the functionality described in the required objectives. However, there are still possibilities of failure depending on various factors. Thus, the project could not be considered as a total success as far as in concerns the performance of the robot. The following paragraphs summarize the results of the final tests evaluating the performance according to the predefined objectives.

1. The robot is able to communicate successfully with the GUI via either Bluetooth or USB connection. Both of these options are fully-functional allowing the user to choose between them by selecting the appropriate combo box in the GUI.

2. The robot is able to lift bricks, using a fork-lift mechanism. The fork-lift design was found to be the most suitable for this purpose, allowing the robot not only to lift bricks, but also move them safely around the frame and place them easily to the ground.

3. The robot is able to navigate inside the Frame using two Ultrasonic and one Color sensor. The initial objectives described the robot to use a Compass sensor instead of the second Ultrasonic sensor. However, due to the fact that the Compass sensor was found inappropriate to be used, the plan was altered. The functionality though remained the same, with the robot performing the same tasks in a different way.

4. The robot is NOT able to place the bricks precisely to the required position. The algorithm designed to make the robot move parallel to the West wall does not offer an accurate enough movement. Thus, some bricks may not be placed entirely inside the squares. This event may lead to further implications especially if the squares to be visited will be next to each other. As a result, a demonstration would work best if the selected squares that belong to the same line have at least one space between them.

5. The Color sensor used to recognize specific positions has a slight possibility to fail during the procedure. As a result, the robot may either lose the dispatch point or bump to the wall. In case of such an event, the robot is not designed to apply correction movements. Thus, the procedure will fail unless it is manually put back in line.

4.3 Conclusions

A number of tests were conducted to determine the accuracy of the final application. A set of conclusions were made both for the Interface and the Robot. The tests provided the opportunity to evaluate the project and compare it against the project specification requirements. The following chapter summarizes the main outcomes of the project implementation.

Chapter 5

Conclusion

This chapter summarises the main outcomes and conclusions resulting from this body of work.

5.1 Conclusions

During the project implementation a number of problems arise mostly related with the NXT sensors. Either mechanical malfunctions or lack of programming experience in specific tasks, created the need to search for alternative solutions. The continuous testing and experimentation with the robot's behavior provided the right inspirations, in time, to overcome the difficulties. In addition, dividing the project in separate parts made it easier to work with short activities, identifying problems faster.

Testing the final application gave the opportunity to evaluate the project and led to the following conclusion. The robot, despite the necessary changes, behaves as it was described in the project specification. However, the brick allocation is not extremely accurate. It is quite unlikely for a brick to be placed entirely inside the square, for reasons described in the evaluation of the process. Below, follows a list of the achievements regarding the project specification requirements.

Graphical User Interface

- Provides a Grid to allow the user draw specific patterns
- Achieves communication with the robot either by BT or USB

The robot

- Operates inside a limited environment.
- Achieves movement using Motors and balancing wheels
- Navigates using Ultrasonic and Color sensors
- Lifts bricks from a specific location
- Places bricks relatively close to positions provided by the user

5.2 Future Work

During the project implementation a number of possible adjustments were discovered. Some of them could improve existed functionality, while others could add some extra. However, since there was no efficient time to develop them, a small description was written to record them for the future work. The improvements include the following:

5.2.1 Autonomous Brick Dispenser

The current project does not deploy a particular mechanism to move the bricks to the desired position. The bricks have to be placed there by hand. Each time a brick is lifted by the robot, a second one should be moved in. However, that means that the project is no completely autonomous. A solution to this issue would be to use a second robot to perform this activity. Robot number 2 would have to be attached to a factory-ribbon mechanism via a motor. The robot would be able to control the ribbon by simply moving the motor. The ribbon itself would have to be placed in front of the dispatch point. Thus, the bricks would slide on the ribbon one by one until located to the dispatch position.

There are two possible ways for the robot to control the flow of bricks to the drop point. The first one would demand the two robots to be connected together via Bluetooth technology and communicate with each other the same way the robot now communicates with the GUI. By the time the Master would be in need for a brick, a command would inform the Slave to move the ribbon for a specific distance in order for one to reach the

drop point. A second solution would be to attach either a color or an ultrasonic sensor directly to the side of the dispatch point. Both sensors would be capable of recognizing if there is a brick in front of them or not. Thus, if the dispatcher would be empty, the robot would move the ribbon and dispatch another brick.

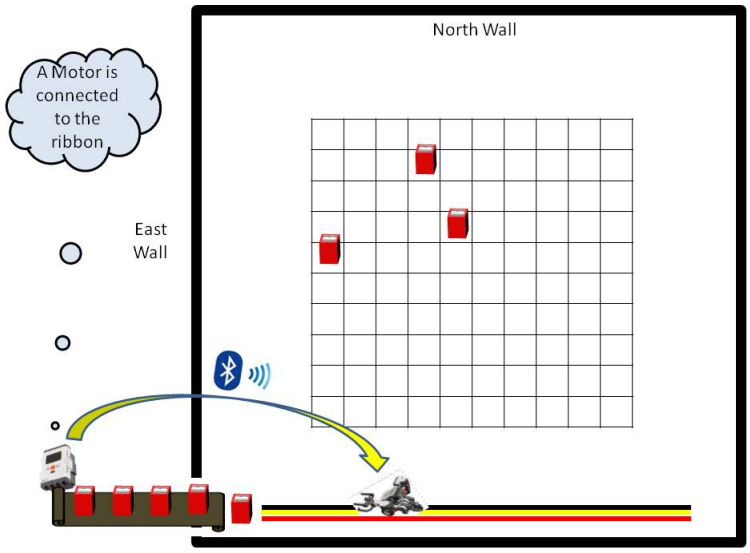


Figure 5.1: Autonomous Brick dispenser.

5.2.2 Building a 3D Pattern

Currently, the robot has the ability to design a 2D pattern on the Grid. Adding a second layer of bricks is not possible at the moment. However, this functionality would not be far away. In order for a brick to be placed upon another, the robot would have to approach the position very accurately. The placement would be the same activity as before with the only difference that the fork-lift would raise double the height than previously. But how could the robot ensure a really accurate movement towards the desired position?

During the implementation of the project, it was discovered that the compass sensor was not functioning properly. Thus, using it was quite impossible. However, a new compass may be available to the future. Combining the Ultrasonic sensors with the Compass would offer the best solution for the above problem. Such combination would result to a much more stable, accurate movement. As a conclusion, placing one brick upon another would be a matter of the same movement which would just occur twice. The result would be a 3 dimensional pattern.

5.2.3 Multiple Builders

The most ambitious improvement to be made for the project would be to add multiple builder robots to design the pattern. A swarm of robots would exchange information with each other in order to build the design as a team [9], [10]. However, such improvement would probably have to change the project entirely since modifications would have to be made to the core of the code. Though challenging, it would be a complicated procedure, demanding a lot of new functionality. The robots would have to be able to perform sophisticated movements in order to avoid each other inside the frame. In addition, it would be necessary to divide the pattern between each other since each place would have to be visited by only one robot at a time. As a conclusion, it is quite clear that the task would be a whole new project on its own. However, it would open new horizons for the NXT robot programming.

Bibliography

[1] Author U. Lego Digital Designer; 2014. Available from: http://ldd.lego.com/en-gb/ [cited 30/8/2104].

[2] Golovine JC. Interactive 2D Graphics; 2014.

[3] Sink J. Development of a prototype Lego NXT location tracking robot. Gahanna Lincoln, Ohio; 2011.

[4] Author U. Compass Sensor Failure; 2014. Available from: http://ftcforum.usfirst.org/showthread.php?1639-Experience-Programming-with-Compass-Sensor&s=9531a8acc2bd459e3bc69ccf0b525606 [cited 16/8/2014].

[5] Author U. Ultrasonic Advantages and Disadvantages; 2014. Available from: http://www.ab.com/en/epub/catalogs/12772/6543185/12041221/12041229/Ultrasonic-Advantages-and-Disadvantages.html [cited 12/8/2014].

[6] Author U. Ultrasonic Sensing; 2014. Available from: http://www.ab.com/en/epub/catalogs/12772/6543185/12041221/12041229/print.html [cited 11/8/2014].

[7] Zamora-Martinez F. Line Follower using Q-Learning. F. Zamora-Martinez; 2013. Available from: http://cafre.dsic.upv.es:8080/~pako/legos.html [cited 14/8/2014].

[8] Bracher S. Fuzzy Line Following NXT Robot; 2014. Available from: http://stefans-robots.net/en/fuzzy-line-follwing-nxt-robot.php.

[9] Stetsyuk M. System Design For Coordinated Swarming Via Communication And Navigation Using LEGO NXT Mindstorms Robots; 2013. Available from: http://maksym-stetsyuk.tumblr.com/.

[10] Walters R. Real life Constructicon quadcopter robots being developed; 2014. Available from: http://www.extremetech.com/extreme/107217-real-life-constructicon-quadcopter-robots-being-developed [cited 31/8/2014].

Appendix A

Project Specification

The aim of this project is to design a robot that will be able to construct a specific 2D-geometric pattern using LEGO bricks. The desirable pattern will be drawn by the user on a Java GUI application and then will be uploaded to the brick via USB or Bluetooth technology. The robot to be used will be a NTX 2.0 brick and the model will be designed to be capable of lifting bricks and move them around the area. It will operate inside a wall frame (2.5 X 2.5 m) using Ultrasonic and compass navigation techniques. The picture below depicts the prototype design drawn by hand (Figure A.1). It is the initial idea used to describe this functionality.

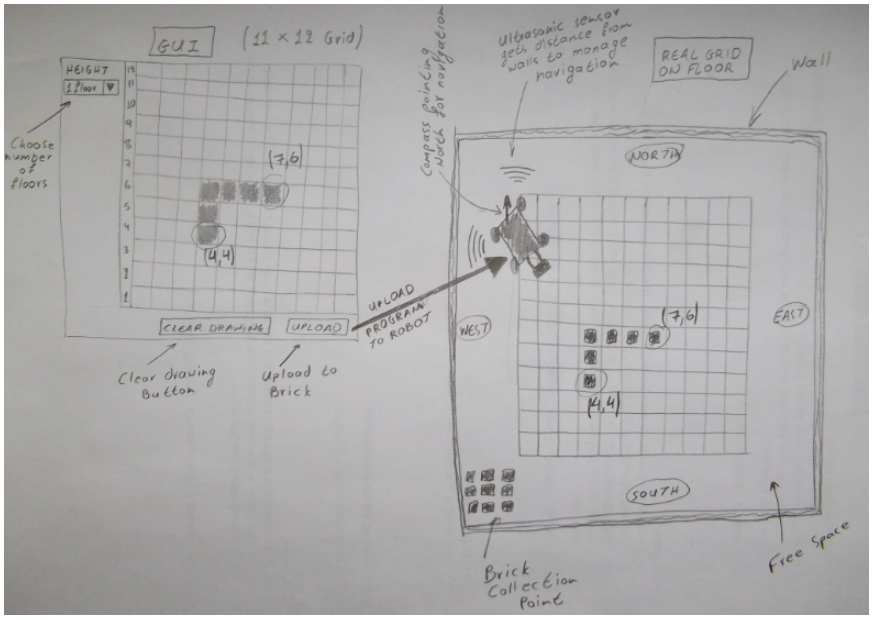


Figure A.1: Prototype design depicting the GUI (left) and the operating frame (right)

A.1 Pseudocode

Below is the prototype pseudocode to be used for managing the whole operation. Be aware that it will most likely involve some changes by the time the project starts.

User:

Algorithm 9: User Procedure Pseudocode

1. Select from the GUI the grid-squares that need to be filled with bricks.
if *If the drawing needs to be cleared* **then**
    click CLEAR button.;
else
    Continue
end
2. Click UPLOAD button to upload the command to the robot.

Robot:

Algorithm 10: Robot Procedure Pseudocode

1.Receive command from the GUI. The command includes the (X,Y) coordinates of each brick to be placed.;
while *There are more bricks to be placed* **do**
    2.Take position to the brick collection point.(It is possible that it will be the starting point);
    3.Pick-up a brick from the collection point;
    4.Uss the compass to create a sense of where the front-part of the area is and where the back-part is. North wall-side will be the benchmark for the navigation;
    5.Use the Ultrasonic sensor to find the distance from the two sides of the wall;
    6.Read the first pair of (X,Y) coordinates which indicate X cm distance from the 1st side and Y cm distance from 2nd side.;
    if *Distance is not correct* **then**
        7.Move around the grid until X,Y distance from the wall sides is correct.;
    else
        Stay at this place
    end
    8.Put the brick to the appropriate square;
    9.Return to collection point. (Probably the starting point);
end

A.2 Minimum Functional Requirements

The following functional requirements will be incorporated into the final project. These requirements detail the basic functionality that the robot will be able to perform. However, there may be additional functions delivered on the final project.

The GUI:

1. Will provide a Grid panel to the user allowing him to draw a specific pattern. The Grid will be scaled down to the real Grid drawn on the floor.

2. Will provide some buttons to upload the drawn design to the brick and begin the building process (Figure A.2).

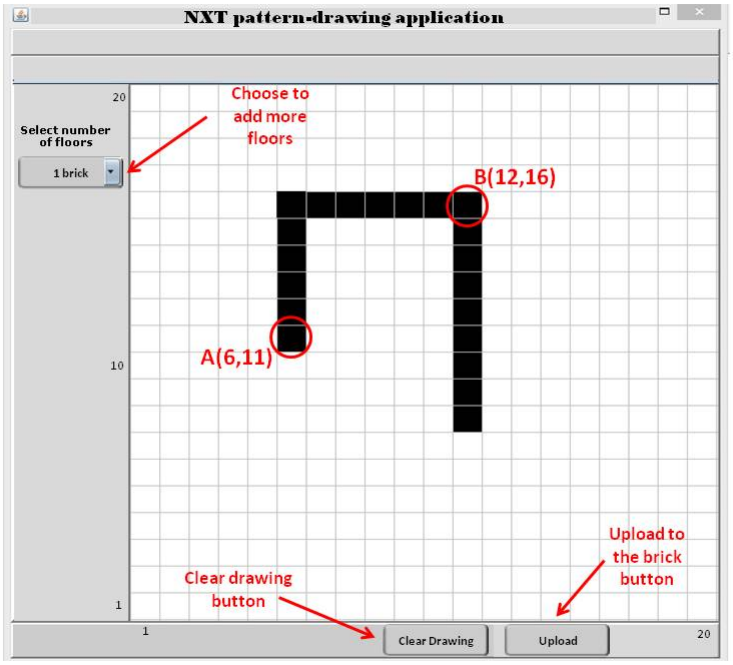


Figure A.2: Preliminary GUI for PC control

The operating range:

1. The robot will operate inside a wall frame (probably wooden 2.5 X 2.5 m). Average height approximately 20cm.

2. A Grid (2X2 m) will be designed on the floor to demonstrate the precision of the robot. The size of the square blocks will be enough to host the bricks. Only one brick will fit inside each block (Figure A.3).

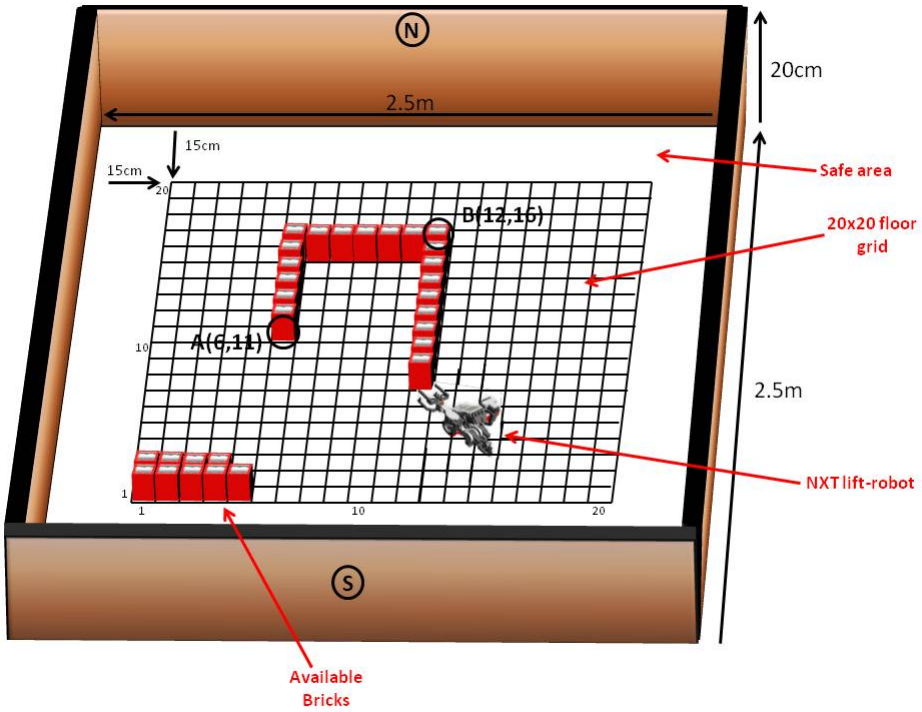


Figure A.3: The picture depicts the robot's operational environment

The user:

1. Will be able to draw a specific 2D-geometric pattern using the GUI.

2. The pattern will be flexible depending on the user requirements, but only one will be constructed each time the application runs.

The robot:

1. Will operate inside the wall frame.

2. Will handle the movement using either wheels or treads.

3. Will navigate precisely inside the frame using Ultrasonic and compass sensors.

4. Will be able to locate and lift specially designed LEGO bricks.

5. Will be able to move the bricks around the frame and finally place them on the specific same grid blocks entered on the GUI.

A.3 Additional Functional Requirements

Additional functional requirements are some extended statements that are hoped to be implemented to the final project. However, it cannot be guaranteed that they will be achieved by the end of the deadline.

1. The user will be allowed to add a second and third line of blocks above the already existed ones. This could end up looking like a 3D wall pattern.
2. The user will be allowed to choose a color for each brick.
3. The bricks will be divided into teams of different colors.
4. The robot will be able to distinguish the color of the bricks, using color sensors, and place them according to the drawn pattern.

A.4 Design Objectives

1. The GUI will be simple to use.
2. The drawn pattern will be uploaded to the brick via Bluetooth. In case of unexpected behavior, alternative options will be provided (connection via USB cable or direct manipulation of the source code).
3. The robot will be able to operate without any further human interaction until the construction has reached to an end.

A.5 Design Goals

1. The Graphical User Interface (GUI) will be easy to use. It will depict a scaled replica of the real grid designed on the floor.
2. The Graphical User Interface (GUI) will be able to transfer the coordinates from the computer to the NXT brick via Bluetooth.
3. There will be alternative options to transfer the data (USB connection) in case of a communication failure.
4. By the time the data will have been transferred to the brick, no more human interaction will be necessary until the process has been completed.

Appendix B

Project Management

Figure B.1 depicts the project plan as it was designed prior to the implementation phase. The plan offered the opportunity to define various tasks either related to the project or not. The project was going on along with other responsibilities, thus other events may had to interfere during the implementation. Recording most of these activities helped to create a better understanding of their significance and prioritize them using story points. It also created a sort of template to work on and use as a guide during the implementation.

The plan was followed quite accurately without many implications since the last semester was mostly focused on the project. However, not everything run according to the schedule. A few changes were applied regarding the tasks that involved programming. Such tasks were quite difficult to be defined by due dates since going back and forth for changes, or working several of them in parallel was often. This is the main reason why the plan contains only the most important, scheduled dates while the remaining were left Open.

No	Project Implementation Tasks	Story Points	Due by	Notes	Priority
1	Create Prototype Designs	30	10/6/14	Designs that describe the procedure. Make a plan and examine possible alternatives	Medium
2	Prototype GUI Classes design	20	10/6/14	Design the basic architecture of the GUI classes	High
3	Graphical User Interface part-1	40	OPEN	Program the main functionality (Draw methods, Translations)	High
4	Graphical User Interface part-2	50	OPEN	Acheive communication with the robot. Make sure data can pass via USB and BT. At this point what kind of data does not matter.	Very High
5	Robot Model design	20	OPEN	Build a robot capable to host required sensors and achieve movement	Medium
6	Test sensors	40	20/6/14	Test desired sensors individualy. Make sure they fit for purpose	Very high
7	Prototype Robot Classes design	20	OPEN	Design the basic architecture of the robot classes	High
8	Robot programming	40	OPEN	Program the main functionality (Movement, sensors)	High
9	Robot Experiments	30	31/7/14	Since there is a functional robot examine its behavior. Change various factors. Test sensors as a team	Very High
10	Construct the Frame	10	OPEN	Boards, walls, grid. Construct the frame.	Low
11	Test the Frame	20	OPEN	Test robot's behavior inside the frame. Changes to the materials may be essential. Especialy the walls.	Medium
12	Evaluation and Testing	30	20/8/14	Run a number of tests. Each test should include a new use case scenario starting the procedure from the begining. Test many	High
13	Design the report Structure	15	OPEN	Collect reports of projects, Create the basic template	High
14	Populate the report	10	OPEN	Collect all drafts and insert them into the report	Medium
15	Intranet systems Assesed Lab		11/8/14		
16	Intranet systems Exams		18/8/14		
17	Final arrangements	20	28/8/14	Move frame to the university. Test behavior there. Apply possible changes needed	Very High
18	Interview		27/8/14		
19	Presentation	10	OPEN	Writte the presentation of the project	Low
20	Last check	10	30/8/14	Check if report is ready, spelling, details, signatures etc	Medium

Figure B.1: Project Plan

Appendix C

GUI User Manual

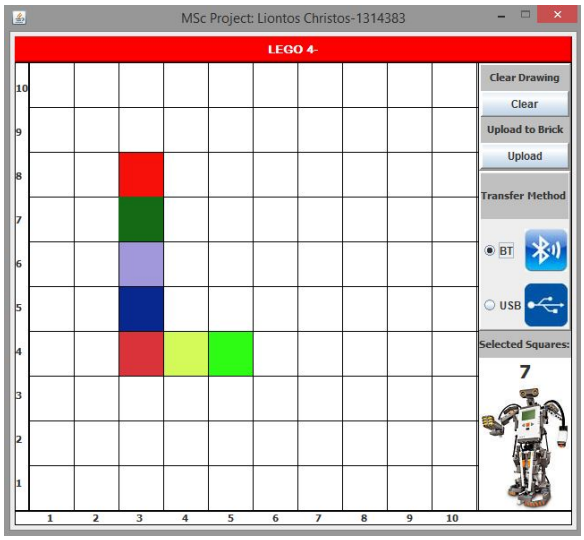


Figure C.1: Graphical User Interface

The Graphical User Interface (Figure C.1) was designed to allow communication between the User and the Robot. The functionality it offers is limited, making it very easy to be used by any kind of user, experienced or not. There are four different buttons appeared:

1.The Clear button. Erases any current design from the Grid
2.The Upload button. Initiates the transfer procedure
3.The BT and USB radio buttons. Only one can be selected at a time, switching between Bluetooth and USB communication

To initiate a new project:

1.Place the robot in front of the Dispatch Point
2.Switch the robot on
3.Navigate through the programs menu and run the building one
4.Select transfer method in LCD screen
5.Run the GUI from your computer
6.Select the desired squares in the Grid
7.Select transfer method
8.Upload the program to the brick

Appendix D

Frame structure

The robot's operating range was supposed to be a square frame designed by four foam walls. However, during the implementation of the project it was decided that only two walls were necessary for the robot to navigate. Thus, the final structure includes only the West and South walls. A list follows including the materials used to build the frame. Further below there are a few pictures taken during the building process. The frame was constructed in such way in order to be as portable as possible.

1. A wooden exoskeleton to support the walls

2. Foam walls (four pieces)

3. Tin foil to dress up the walls

4. Thick paper for the Grid floor

5. Tapes and Markers to paint the shapes

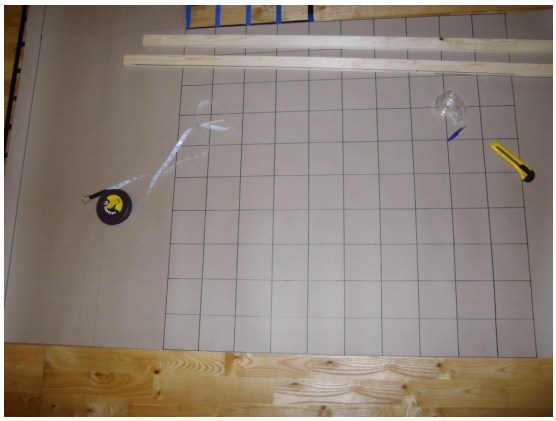


Figure D.1: The Grid is drawn on a piece of thick paper

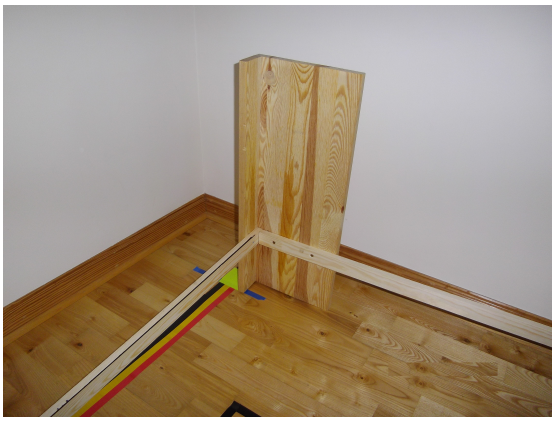


Figure D.2: Wooden corner supports the exoskeleton

Figure D.3: Exoskeleton supports the walls

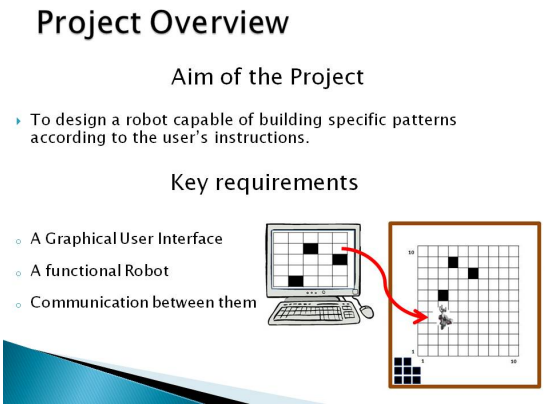
Figure D.4: Foam walls dressed with tin-foil

Figure D.5: The frame completed

Appendix E

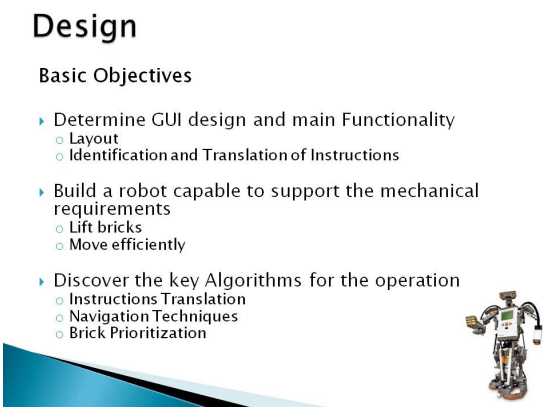
Presentation Slides

Slide 1



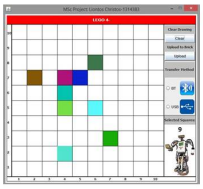
Slide 2

Slide 3



Slide 4

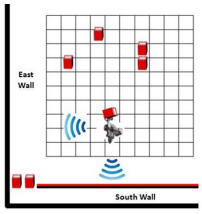
Implementation



- Graphical User Interface Front End Functionality
  - Draw Selected Squares
  - Clear Pattern
  - Upload Data either via USB or BT technologies
  - Count Selected Squares
- Graphical User Interface Back End Functionality
  - Identification of Selected Squares
  - Translation into pair of Coordinates
  - Data transfer to the Brick

Slide 5

Implementation



- Robot Functionality
  - Receive Data from Interface via USB or BT technologies
  - Translate Coordinates into Distances
  - Lift & Move Bricks using the Fork-Lift operator
  - Navigate using Echolocation Techniques
  - Follow lines and Recognize Colored Positions
  - Replicate Drawn Designs

Slide 6

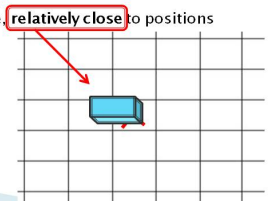
Testing

- Testing individual activities one by one
  - Communications
  - Lift bricks
  - Navigate using Compass and Ultrasonic sensors
  - Navigate using Ultrasonic and Color sensors
  - Maintaining direction
  - Following lines
- Testing the entire application
  - Combine separate activities into a Universal one
  - Test loop behavior for more that one brick placement

Slide 7

Evaluation & Conclusion

- Graphical User Interface that
  - Is simple to use
  - Communicates successfully with the robot
- Robot that
  - Navigates using US and Color sensors
  - Follows lines
  - Uses Echolocation to navigate
  - Places several bricks at a time, **relatively close** to positions provided by the user



Slide 8

Appendix F

Project Log

The following is a weekly summary of the work carried during the project implementation phase. It covers tasks that were completed, reviews of meetings held with project supervisor and further research.

Week ending on 13th June
Meeting with the supervisor:YES
Summary of discussions:
1. General discussion over the start project (aim, objectives, planning, meetings, structure).
2. Instructions given according to the structure of the project.
3. Discussion about the initial plans provided.
4.Emphasis on the project report template.
Description of progress this week:
1. Main plan was created
2. A brief pseudocode was created regarding the robot's functionality (handwriting)
3. Communication between GUI and robot was tested, though failed.
4. Initial robot model was designed, though may change in future.

Week ending on 20th June
Meeting with the supervisor:YES
Summary of discussions:
1. Solutions Proposed
2. Alternative plans proposed
3. Main GUI description discussed
4. Project plan

Description of progress this week:

1. Communication between robot and GUI established
2. Communication functionality and selection between BT and USB developed
3. Report template was created
4. GUI description added
5. Ultrasonic sensor tests added
6. Frame materials found and measured

Week ending on 27th June

Meeting with the supervisor:YES

Summary of discussions:

1. Discussion over the navigation algorithm
2. Auto Correction Route explained
3. Additional objectives proposed
4. Videos

Description of progress this week:

1. Auto Correction Route algorithm
2. Navigation tests
3. Compass faults discovered
4. Approaching the first square method

Week ending on 4th July

Meeting with the supervisor:YES

Summary of discussions:

1. Advice given for the upcoming weeks

Description of progress this week:

1. Compass problems resolved. Compass use was canceled. Written documentation.
2. Alternative found using Ultrasonic and Color Sensor
3. General algorithm created

Week ending on 11th July

Meeting with the supervisor:NO

Description of progress this week:

1. Classes were redesigned using threads
2. Ultra Sonic sensor malfunctions identified
3. Coordinate translation
4. Report

Week ending on 18th July

Meeting with the supervisor:YES

Summary of discussions:

1. Current progress discussed
2. Software problems and proposed solutions
3. Small demonstration of the robot in action (videos)

Description of progress this week:

1. Report
2. Colour sensor testing and evaluation
3. Prototype Colour sensor functionality programmed (line follower)
4. GUI bugs have been corrected
5. New Ultrasonic sensor received

Week ending on 24th July

Meeting with the supervisor:NO

Description of progress this week:

1. Report
2. Ultrasonic sensor tests (positive)
3. Code functionality
4. Frame building (needs adjustments. Not ready to be tested

Week ending on 1st August

Meeting with the supervisor:YES

Summary of discussions:

1. Additional functionality proposals
2. Project report structure

Description of progress this week:

1. Report
2. Code functionality, validation
3. Color sensor adjustments

Week ending on 7th August

Meeting with the supervisor:NO

Description of progress this week:

1. Report. Drafts were imported properly. Design, Implementation chapters populated

Week ending on 15th August

Meeting with the supervisor:YES

Summary of discussions:

1.Report-feedback discussed

Description of progress this week:

1. Report

2. Small adjustments in the code

Week ending on 22nd August

Meeting with the supervisor:YES

Summary of discussions:

1. Reports were read and feedback was given for last changes

2. Some further research proposed

Description of progress this week:

1. Evaluation and Testing final version of the application

2. Finalize report

2. Preparing the presentation

Week ending on 22nd August

Meeting with the supervisor:YES

Summary of discussions:

1. Final adjustments discussed

2. Advice provided

2. Information about presentation and demo **Description of progress this week:**

1. Finalize Presentation