Building a Bluetooth Automatic Obstacle-Avoidance Robotic based in an Agile development process

Karim Maher

Course of Study: M.Sc. INFOTECH

Examiner: Prof. Dr. Stefan Wagner

Supervisor: Ph.D Yang Wang

Commenced: 1. February 2016

Completed: 2. August 2016

CR-Classification: D.2
Abstract

...Nowadays, the selection of the SW development process is the first decision that should be taken before any SW project implementation. waterfall and V-model have been used for many decades, and have shown many improvements in the SW development process. These two methods have also some drawbacks regarding their inflexibility to the changing requirements and their long test cycle time. In addition, the lack of safety analysis techniques has result in an unsafe SW production.

The purpose of this Master thesis is to investigate the effects of applying the agile development process (solo scrum) integrated with the results of safety analysis on the ODA robot (embedded system) development process.

The ODA algorithm has been developed based on the agile development process (scrum) practices like TDD, scrum board, and other practices. For safety consideration, the ODA algorithm has been continuously changed according to the safety requirements. Moreover, a GUI has been implemented so that robot be remotely controlled via Bluetooth and which provides a continuous feedback from the robot.
Contents

1 Introduction 13
  1.1 Scope .............................................................................. 13
  1.2 Motivation ...................................................................... 14
  1.3 Purpose .......................................................................... 14
  1.4 Contribution .................................................................. 14
  1.5 Related works ............................................................ 15

2 Background 17
  2.1 Lego robot with Java development module .................... 17
    2.1.1 Lego NXT ................................................................. 17
      2.1.1.1 Lego Basic Motors and Sensors(HW Structure) ..... 20
      2.1.1.2 Lego Programming Languages(SW Structure) ..... 22
    2.1.2 Lejos ........................................................................ 24
      2.1.2.1 Advantages and Drawbacks of using leJOS Firmware . 24
      2.1.2.2 Programming Process ......................................... 25
  2.2 Agile methods ................................................................. 28
    2.2.1 General Agile methods(XP, Scrum, FDD...) .................. 28
    2.2.2 Scrum development process ....................................... 31
      2.2.2.1 Scrum Theory ...................................................... 31
      2.2.2.2 Scrum Events ..................................................... 32
      2.2.2.3 Scrum Team ....................................................... 34
      2.2.2.4 Scrum Artifacts .................................................. 35
    2.2.3 XP Practice (Test Driven Development) ..................... 36
      2.2.3.1 Test Driven Development versus Traditional Debug Later 36
      2.2.3.2 Test-driven development cycle ................................ 38
      2.2.3.3 TDD Benefits .................................................... 40
      2.2.3.4 TDD Benefits for Embedded Systems .................... 40
      2.2.3.5 Test Doubles ...................................................... 41
  2.3 Safety .............................................................................. 44
    2.3.1 General safety standards, development process, and techniques 44
      2.3.1.1 IEC 61508 and ISO 26262 ................................... 45
      2.3.1.2 FMEA, FTA, STPA ........................................... 49
<table>
<thead>
<tr>
<th>3</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>ODA Control Structure</td>
</tr>
<tr>
<td>3.2</td>
<td>Developing ODA with TDD</td>
</tr>
<tr>
<td>3.3</td>
<td>Improved ODA Control Structure</td>
</tr>
<tr>
<td>3.4</td>
<td>software Safety Verification</td>
</tr>
<tr>
<td>3.5</td>
<td>Final Product</td>
</tr>
<tr>
<td>4</td>
<td>Development process</td>
</tr>
<tr>
<td>4.1</td>
<td>The first sprint</td>
</tr>
<tr>
<td>4.2</td>
<td>The second sprint</td>
</tr>
<tr>
<td>4.3</td>
<td>The third sprint</td>
</tr>
<tr>
<td>4.4</td>
<td>The fourth sprint</td>
</tr>
<tr>
<td>5</td>
<td>Limitations</td>
</tr>
<tr>
<td>5.1</td>
<td>Solo Scrum is not a standard Scrum</td>
</tr>
<tr>
<td>5.2</td>
<td>functions are ideal</td>
</tr>
<tr>
<td>5.3</td>
<td>Bluetooth Communication</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion and future work</td>
</tr>
<tr>
<td>7</td>
<td>Appendix</td>
</tr>
<tr>
<td></td>
<td>List of Abbreviations</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
</tr>
</tbody>
</table>
List of Figures

2.1 NXT Brick .......................... 18
2.2 Light Sensor ........................ 20
2.3 Sound Sensor ........................ 20
2.4 Touch Sensor ........................ 21
2.5 Ultrasonic Sensor .................... 21
2.6 Motor .............................. 22
2.7 Waterfall Model ........................ 28
2.8 Agile Model .......................... 32
2.9 Debug Later Programming ............ 37
2.10 TDD Cycle Time ...................... 37
2.11 TDD cycle ........................... 38
2.12 Isolating dependencies ............... 41
2.13 Types of Test Double ................. 42
2.14 IEC 61508 Life Cycle ............... 45
2.15 ASIL determination formula .......... 47

3.1 Ultrasonic sensor Idea [Gan15] .............. 58
3.2 Obstacle Avoidance Detection Algorithm State Diagram .... 59
3.3 Obstacle Avoidance Detection Algorithm Flowchart .......... 60
3.4 Move Forward Test Case .................. 62
3.5 Move Right Test Case .................... 63
3.6 Lego GUI ................................ 65
3.7 GUI Diagram ........................... 66
3.8 GUI Flowchart .......................... 67
3.9 Forward Method after STPA ............... 69
3.10 Stop Method after STPA ................ 70
3.11 Lego Parking .......................... 71

4.1 Scrum Board ........................... 73
4.2 ODA Lego Robot ....................... 75
4.3 Lejos Motor Isolating dependencies .......... 77
4.4 Lejos Ultrasonic Sensor Isolating dependencies .......... 78
4.5 Robot with rotating Ultrasonic Sensor .......... 84
List of Tables

2.1 Hardware Description .................................................. 19
2.2 Usable Lego Programming Languages .................................. 23
2.3 Basic Lejos NXJ packages ................................................ 27
2.4 IEC 61508 Safety Integrity Level ....................................... 46
## Listings

4.1 Java Junit Test Framework importing ........................................... 75
4.2 Assertions importing ................................................................. 76
4.3 MotorsMoveForwardWhenNoObstacle Test Case .............................. 76
4.4 Forward Method in CUT .............................................................. 76
4.5 Three Motors instantiation .......................................................... 77
4.6 Fake setDistance Method within Fake Ultrasonic sensor Class ........ 78
4.7 MotorsStopWhenDetectingObstacles Test Case .............................. 79
4.8 Stop Method within CUT ............................................................ 79
4.9 Fake stop and isMoving Methods within FakeMotor Class .............. 79
4.10 Forward and Stop methods within CUT ......................................... 80
4.11 GUI Right Button ................................................................. 81
4.12 GUI Left Button ................................................................. 81
4.13 GUI Forward Button ............................................................. 81
4.14 GUI Backward Button ........................................................... 82
4.15 GUI Stop Button ............................................................ 82
4.16 GUI "+" Button ............................................................. 82
4.17 GUI "." Button ............................................................. 82
4.18 MoveRightWhenNoObstacleOntheRight Test Case ....................... 84
4.19 Edited Stop method after rotating Ultrasonic sensor ..................... 85
4.20 Fake method isRightMoving within Fake motor class .................. 86
4.21 MoveLeftWhenNoObstacleOntheLeft Test Case .......................... 86
4.22 UltraSonicSensorMotorSecondAngle Test Case .......................... 88
4.23 ODA GUI Button .......................................................... 93
4.24 GUI Distance to Obstacle worker thread ................................... 96
4.25 GUI Motor Speed worker thread ............................................. 97

7.1 Obstacle Avoidance Detection Code after STPA results ................. 103
7.2 Obstacle Avoidance Detection Code before STPA results ................. 107
7.3 Obstacle Avoidance Detection Test Cases before STPA results ........ 110
7.4 BasicMotor class(FakeMotor) .................................................. 112
7.5 Motor class ............................................................................ 114
7.6 UltrasonicSensor(Fake UltrasonicSensor class) .............................. 115
7.7 CUT before STPA ................................................................. 116
<table>
<thead>
<tr>
<th>Listings</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.8 GUI code</td>
<td>119</td>
</tr>
<tr>
<td>7.9 Forward Method Before Safety Analysis</td>
<td>127</td>
</tr>
<tr>
<td>7.10 Forward Method After Safety Analysis</td>
<td>127</td>
</tr>
<tr>
<td>7.11 Stop Method before STPA</td>
<td>128</td>
</tr>
<tr>
<td>7.12 Stop Method After STPA</td>
<td>128</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Scope

The general scope of my master thesis is to develop the ODA function (Embedded system) in an Agile method.

1. HW environment
   LEGO® MINDSTORMS™ NXT is selected as the HW environment. The control unit and power supply is NXT brick, which is connected with motors and ultrasonic sensors to detect the obstacles. LEGO® MINDSTORMS™ NXT is based on ARM9 processor with embedded Linux operating system, and Bluetooth wireless communication block.

2. SW environment
   A Personnel Computer (PC) Graphical User Interface (GUI) is developed in Java to run the Robot in the Obstacle Detection and Avoidance mode by using an Ultrasonic sensor in front of the Robot which continuously track the Distance to Obstacle and update the GUI with this value, the Obstacle Detection and Avoidance mode can be interrupted so that the Robot’s Direction (Right, Left, Forward, Backward) could be controlled from PC via Bluetooth, we neglect the hardware related development and directly use LEGO® MINDSTORMS™ NXT to achieve the desired operation. We separate the final function into three sprints during Scrum development process. Each function is acted as a simulated customer requirement. After each sprint, we realize one of function and review the product.

3. Agile technique
   Test Driven Development (TDD) is used in the Software Development, in which Test Cases are written before Software Development, the real implementation is swapped with something called Test Doubles which have a simplified behavior and which allow the code to be tested independently on the LEGO Robot Hardware, Fake Object which is a type of Test Doubles is used to emulates the Real Application.
1 Introduction

1.2 Motivation

The ability to avoid obstacles automatically is an important part of intelligent robotic vehicles, and creates a safety assurance for future travel. Obstacle Avoidance is the lowest, or zeroth, level of competence, meaning it is the core functionality of a mobile robot system upon which everything else depends. If a robot can be made to avoid coming into contact with objects in the environment, then other higher-level capabilities can safely be incorporated into the system. An ultrasonic sensor would be the most suitable for obstacle detection.

The sequential design process Waterfall method has many problems regarding flexibility because it is difficult to go back and change something that was not well-thought out in the early stages of Development, the unavailability of the produced software until late during the life cycle which result in a lack of information for the customers on the progress of the product under development. All these problems lead us to navigate to the Iterative Agile Development process during Software Development, and which uses the TDD where unit testing is repeatedly done on source code in an iterative way and in which Testing is done before Development.

1.3 Purpose

The goal of my Master Thesis is to investigate the effects of applying the Agile Development Process and techniques on the Embedded Software Development Process of LEGO Mindstorms NXT Robots.

1.4 Contribution

1. Realizing the automatic Obstacle Detection Avoidance (ODA) in Lego Robot and its communication.

2. Adopting an agile development process “Solo Scrum” to develop an Embedded system.

3. Realizing the safety-critical functions development in an iterative development process.
1.5 Related works

The ODA is usually being implemented in other Master theses projects by using the Waterfall development process or the V-Model. There is currently no national standards exist which are directly applicable to the design, construction, operation and maintenance of Autonomous Mobile Robots, but there is a Standard for Obstacle Avoidance Detection called ANSI/ITSDF B56.5 Developed and approved by the American National Standards Institute, this standard is a guide to safety requirements for system suppliers, manufacturers, buyers and users for the design, construction, application, operation and maintenance of unmanned guided industrial vehicles and automated functions of manned industrial vehicles[NY]. American National Standards Institute (ANSI)/Industrial Truck Standards Development Foundation (ITSDF) B56.5 standard verifies whether or not a system is able to detect an agent that temporarily enters this path and avoid colliding with it.

The Agile techniques is one of the mostly used worldwide methodologies for software development process. Companies that have totally used Agile methodology for their Software projects "have seen great improvements in time to market, product quality and customer satisfaction if it is used in the correct way". Many companies are now looking to not use the Agile process for large projects that contain large number of employees. Experts noticed that firms often apply Agile to medium-sized projects, since they have considerable advantages over larger projects in terms of planning and efficiency. The projects that most benefit from Agile are web technology, mobile technology and the Internet of Things. One of the biggest impacts will be in the testing community, the complexity of testing will increase by orders of magnitude and there will be a strong push to much more test automation to be able to cope.

There is currently no publications on applying the Agile into Obstacle Avoidance Detection using Lejos, but there is only an implemented project published on the Internet which applied TDD with Lejos [jta]. There are some publications on how to apply the Agile Scrum on the Software development process on other functions rather than the Obstacle Avoidance Detection [Nys11].
2 Background

2.1 Lego robot with Java development module

2.1.1 Lego NXT

Presently, the usage of Robots to perform meaningful tasks is widely common all over the world in many fields ranging from simple home tasks to missions in the outer space [AD12]. For the current technologies present in the world, we are not capable to see robots with the same intelligence of humans, but robots now can perform tasks in place of humans with higher reliability and accuracy. In the future, the state of technology will be evolved so that the Robots can make more complex tasks in place of humans.

A Robot can be defined in two ways: The first definition, is an object that replace the humans. The second definition, it is an object that is moving in a certain place and react to this environment by using sensors, actuators, and motors. The selection of these sensors and actuators is done based on the required application. The Robot need something like human brain to perform a certain algorithm and to synchronize the sensors with actuators used. This brain is quite simply a (here Lego NXT Brick), which take an input signal from sensors and output other control signals to the actuators in response to this input signal, this output could be moving motor direction for example.

We would use Robots to replace humans for many reasons, here are some of them:

- Do continuous and iterative tasks with the same quality of work.
- Do tasks that may be dangerous to humans.
- Reach places that maybe dangerous for humans.
- work for a long period of time without any Human interaction.

LEGO Mindstorms NXT is a robotic kit that could be programmed in several ways, it was produced by Lego in July 2006. The NXT Brick is the main brain used to control the movement and interactions of the Lego with the surrounding environment based on input signals from sensors.
NXT is widely used and there are several resources on the Internet which provides technical support and there are also many ready implemented examples on the Web \cite{Ljfs}. There are many reasons to use the Lego NXT such as:

- The construction of robot is pretty simple. No electrical wiring or soldering is needed to build the robot.
- Can be programmed in different programming languages.
- can be simply expanded by interfacing with different sensors and by connecting to other devices.
- communication capability with other devices via USB, or Bluetooth.
- Can be used with different levels of complexity.
- The availability of several open source firmware.

The major drawbacks of the Lego NXT is

- Hardware Limits.
- computational power.
- hardware expansion capabilities.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{nxt_brick.png}
\caption{NXT Brick}
\end{figure}
In the following table, the technical specifications of the Lego NXT robot are being showed.

<table>
<thead>
<tr>
<th>Motor Ports</th>
<th>The NXT has three output ports (A,B,C) for attaching motors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor ports</td>
<td>The NXT has four input ports (1,2,3,4) for attaching sensors.</td>
</tr>
<tr>
<td>USB port</td>
<td>Used to download programs or Firmware from PC to NXT, Bluetooth can also be used to upload the programs.</td>
</tr>
<tr>
<td>Loudspeaker</td>
<td>Run Programs with real sounds and listen when the program is executing.</td>
</tr>
</tbody>
</table>

**NXT Buttons**

- Orange button: On/Enter/Run
- Light grey arrows: Used to move left and right in the NXT main menu
- Dark grey button: Clear/Go back

**Technical specifications**

- 32-bit ARM7 microcontroller
- 256 Kbytes FLASH, 64 Kbytes RAM
- 8-bit AVR microcontroller
- 4 Kbytes FLASH, 512 Byte RAM
- Bluetooth wireless communication
- USB full speed port
- 4 input ports, 6-wire cable digital platform (One port includes a IEC 61158 Type 4/EN 50 170 compliant expansion port for future use)
- 3 output ports, 6-wire cable digital platform
- 100 x 64 pixel LCD graphical display
- Loudspeaker - 8 kHz sound quality. Sound channel with 8-bit resolution and 2-16 KHz sample rate.
- Power source: 6 AA batteries.

**Table 2.1: Hardware Description**
2 Background

2.1.1.1 Lego Basic Motors and Sensors(HW Structure)

Sensor is one of the most important components used in a robot, the Lego NXT is supplied with a big variety of sensors. The main aim of sensors is to sense the surrounding environment and which will allow robot to interact with its surroundings.

The preceding operation is called autonomy, which mean take decisions independently from external factors or interactions. Our Robot is considered as autonomous Robot since it takes the input signals from its sensors and then the NXT Brick take decisions based on these inputs.

**Light Sensor**
It is used to differentiate between light and darkness, differentiate between light intensity within a room and of coloured material.

![Figure 2.2: Light Sensor](image)

**Sound Sensor**
It is used to measure different sound pressure with the same sensitivity, it can measure up to 90, the sound sensor output values are represented in percentage, the value is in decibel dB or adjusted decibel dBA. The higher the value the louder the sound. For example: silent living room will have a value of 4%, for someone who is talking the range is from 10% to 30%, loud music playing will be from 30% to 100%.

![Figure 2.3: Sound Sensor](image)
2.1 Lego robot with Java development module

**Touch Sensor**
It is used to sense any touch done on it, can be used to make the robot make some actions based on human or non human touches.

![Touch Sensor](image)

**Figure 2.4: Touch Sensor**

**Ultrasonic Sensor**
It is used to detect the obstacles by measuring the distance to these obstacles. This distance provided in centimeters (cm) and in inches, the range of values range from 0-255 centimeter with a precision of +/- 3 . The distance measurement is done by calculating the time taken by a wave to reach an obstacle and to comeback. Obstacles that are small is difficult to detect in contrary to large objects.

![Ultrasonic Sensor](image)

**Figure 2.5: Ultrasonic Sensor**

**Servo Motor**
Motors are used so that the robot can move in different directions. There is a built-in tachometer/rotation sensor that allows an accurate control of the movement of robot. This Tachometer is used to measure the motor rotations in degree or in one rotation cycle (360 °), with an accuracy of +/- 1 °. The Rotation Sensor provided help to set the motor speed. The Motor can be set to run for a limited or unlimited interval of time.
2 Background

2.1.1.2 Lego Programming Languages(SW Structure)

The programs can be downloaded to the Lego via usb or wirelessly by using Bluetooth. Files can also be shared between two NXT bricks via Bluetooth. There could be a communication between up to three Legos wirelessly via Bluetooth.

The original version of the kit provided includes software for writing programs that run on PC. The software is based on LabVIEW and provides a graphical programming language for writing programs and then downloading to the Lego.

**NXT-G**

This is the programming language used originally with the NXT, it is used for basic programming, such as moving motors, mixing sensor inputs, perform some algorithms, and to learn simplified programming structures and flow control.

Here are some advantages and disadvantages of using this software:

**Advantages of NXT-G :**

- NXT-G can be used for many Operating systems such as Windows XP, Windows Vista and Windows 7, and Mac OS
- NXT-G can be used exchange data via usb cable or Bluetooth.
- NXT-G is much easier than the normal programming languages (C, Java) where developer simply use ready made blocks and connect them according to the application.

**Disadvantages of NXT-G:**
2.1 Lego robot with Java development module

- Program size done by NXT-G is larger than programs done by other programming languages like (C, Java).
- Programs take longer time to be uploaded on the Lego NXT Brick than the other programming languages.
- Large programs, causes the NXT-G to crash and sometimes closed unexpectedly.
- NXT-G software often runs slowly on PCs.

**Unofficial Programming Languages**

Despite NXT-G is considered as an easy and friendly way to program the Lego NXT Brick, but there are some applications that may require more advanced programming style by writing lines of code and which provide more control of the robot.

There a lot of unofficial languages that can be used to program Lego Nxt Brick, the majority of these softwares and languages are free and open source. In the following table, a part of the unofficial programming languages that can be used to program the Lego NXT.

<table>
<thead>
<tr>
<th>Name</th>
<th>Language type(s)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruby-nxt</td>
<td>Ruby</td>
<td>Provides low-level access to the NXT via Bluetooth and also some basic high-level functionality.</td>
</tr>
<tr>
<td>NXT++</td>
<td>C++</td>
<td>Allows the control of the NXT directly from any C++ program, in Visual Studio, Windows.</td>
</tr>
<tr>
<td>ROBOTC</td>
<td>C</td>
<td>ROBOTC gives the ability to use C programming language. It has a built-in debugger tools, Mathematical/Trigonometric operations (sine, cosine, tan,... etc), user-friendly auto-complete function built into the interface, and provided with sample programs</td>
</tr>
<tr>
<td>leJOS</td>
<td>Java</td>
<td>A java based system for advanced programmers can handle most sensors, speech recognition and mapping technology. Can be interfaced with the Eclipse IDE or run from the command line. It provides full support to approximately all Operating Systems.</td>
</tr>
</tbody>
</table>

*Table 2.2: Usable Lego Programming Languages*
2 Background

2.1.2 Lejos

In the previous sections, we have seen the majority of programming languages that can be used with Lego robot. Now, we will start with the Lejos Firmware, which is the mostly used firmware with the Lego robot. LeJOS is a Java based firmware which replace the original firmware provided for the Lego Mindstorms bricks. It includes a Java Virtual Machine (JVM), which allows the programming of Lego Mindstorms NXT in Java programming language. The Lejos Firmware can be used to control motors and to deal with sensors, via accessing the I2C ports of the Lego NXT robot.

The implementation details of the LeJos is being concealed behind multiple Object Oriented Programming (OOP) interfaces, which will make the LeJos user to work with high abstraction without going into details in the hexadecimal addresses of the Hardware (HW) components.

Libraries for abstract functions like mapping and navigation is provided in leJOS Firmware. Implementations of some feedback controllers, like Proportional Integral Derivative (PID) controller and Kalman filter noise reduction algorithm is also provided within this firmware.

2.1.2.1 Advantages and Drawbacks of using leJOS Firmware

**Advantages of LeJOS NXJ:** leJOS has many advantages rather than the other programming languages used for the Lego NXT [Mor].

- It uses the industry-standard Java language.
- It provides object-oriented programming.
- It is an open source project.
- It can be used with several Integrated Development Environment such as Eclipse and Netbeans.
- It has cross platform support Windows, Linux, Mac OS X.
- It is much faster than NXT-G.
- It has full support for Bluetooth.
- It allows motor control with high accuracy.
- It supports sensors which are not manufactured by the Lego company or which is not manufactured to work with Lego NXT.
2.1 Lego robot with Java development module

- It supports remote monitoring and tracing of your leJOS NXJ program from the PC.
- It provides Math and trigonometric functions.
- It supports listeners and events.
- It allows garbage collection for memory management.
- USB can be used for debugging.
- It is provided with ready made projects.
- It supports remote execution from the PC or Mobile.
- The forums on Lejos Web site help to solve any problems the developers face.
- It has telerobotics support via standard Transmission Control Protocol (TCP)/Internet Protocol (IP) sockets.
- It supports NXT to NXT Bluetooth communications.
- It supports Bluetooth communication with another devices.

**Lejos drawbacks:**

- limited on board RAM.
- limited on board flash memory for storing programs (Java is not known as compact language).

2.1.2.2 Programming Process

In this section, i will show the used methods during development process, the complete Lejos API could be found on the Lejos website [01].

**Motors**
Three instances of Motor are available: "Motor.A", "Motor.B", and "Motor.C". I will show as mentioned the mostly used methods for the motor class in the Lejos NXJ and which i have used personally during development. The complete motor class methods can be found on the lejos website.

- **void backward()**
  Causes the motor to move backwards.

- **void forward()**
  Causes the motor to move forward.
2 Background

- **int getSpeed()**
  Returns the current motor speed in degrees per second.

- **boolean isMoving()**
  Returns boolean true if the motor is moving.

- **void rotate(int angle)**
  causes motor to rotate through angle.

- **void rotateTo(int limitAngle)**
  causes motor to rotate to limitAngle; Then getTachoCount should be within ± 2 degrees of the limit angle when the method returns

- **void setPower(int power)**
  sets motor power.

- **void setSpeed(int speed)**
  Sets motor speed, in degrees per second; Up to 900 is possible with 8 volts.

- **void stop()**
  Causes motor to stop, pretty much instantaneously.

Basic Sensors

UltraSonic Sensor

- **public UltrasonicSensor(SensorPort port)**
  an instance of Ultrasonic Sensor class is created using this constructor. The "port" variable is the port in which the ultrasonic sensor is attached.

- **public int getDistance()**
  Return distance to an obstacle. It return a value range from 0 to 255. The 255 value is showed if no obstacle is in range.

- **public int off()**
  This instruction will make the Ultrasonic sensor off.

- **public int reset()**
  Perform a soft reset to the ultrasonic sensor, and after this instruction the sensor will operate in continuous mode.
Some of the NXJ packages which will be used during development are shown in the following table. Other packages could be found through the Lejos website [01]

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lejos.nxt.comm</td>
<td>NXT communication classes</td>
</tr>
<tr>
<td>lejos.nxt.remote</td>
<td>Remote NXT access over Bluetooth</td>
</tr>
<tr>
<td>lejos.nxt</td>
<td>Access to NXT sensors, motors, etc.</td>
</tr>
<tr>
<td>java.lang</td>
<td>Core Java classes</td>
</tr>
</tbody>
</table>

Table 2.3: Basic Lejos NXJ packages
2 Background

2.2 Agile methods

2.2.1 General Agile methods (XP, Scrum, FDD...)

The first decision we take for any project implementations is “Which software development process will we use?”. The Software development Process is the way which organizing the work of software development. The two most popular software Development Processes are: Agile and Waterfall.

**Waterfall** is a sequential process of events, these events in order are Analysis, Design, implementation, Testing, Deployment, and Maintenance. Each of these events represents a separate stage of software development, and each stage generally finishes before the next one can begin.

There are positive and negative things about the Waterfall approach.

**Advantages of using Waterfall:**

- The early agreement between customers and developers about the requirements, make the designing and planning more easy and straightforward. so it will be more easily to measure the progress, because the goal of the work is pretty clear from the beginning.

- Waterfall approach make it possible for the team members (developers, testers, business analyst) to work in parallel in different aspect of same project.
• No overlapping between phases.
• works well with smaller and medium sized projects where requirements is already known in the early stages.

Disadvantages of using Waterfall:
• not for medium or large projects.
• It will be difficult to come back to early stages, once we are in the Testing Phase. And if it is possible it will be with very high cost.
• Dissatisfied customers, because sometimes, customers maybe not able to visualize the final software product based on the requirement documents in the early stages which will lead to product that does not match with customers needs.

For all these Drawbacks, i decided to navigate to the Agile Development process.
The are many agile techniques, they share common characteristics and methods. But they are different in some terminologies, strategies, and applications [Pre01].

Here is a summary of the main agile software development processes:

Feature-driven development (FDD): This incremental and iterative software development process mixes the best practices into one approach. It has five basic activities: develop overall model, build feature list, plan by feature, design by feature, and build by feature.

Extreme Programming (XP): Extreme Programming (XP): known as XP, it is a type of software development aimed to improve software quality in response to the continuous changing of the customer requirements.

Adaptive System Development (ASD): Adaptive system development has the idea that a project has to always be in a state of continuous updating and learning from changing project state. ASD has a series of three repeating series: speculate, collaborate, and learn.

Dynamic Systems Development Method (DSDM): it is used to deliver business solutions in a rapid and valuable way. DSDM is based on eight principles: focus on the business need, deliver on time, collaborate, never compromise quality, build incrementally from firm foundations, develop iteratively, communicate continuously and clearly, and demonstrate control.

Lean Software Development. (LSD): Lean Software Development applies the Lean IT principles to software development. It is based on seven principles: eliminate waste,
amplify learning, decide as late as possible, deliver as fast as possible, empower the team, build integrity in, and see the whole.

**Crystal Clear:** used with teams from six to eight developers and it focuses on the people themselves, and not processes. It requires: frequent delivery of usable code to users, reflective improvement, and osmotic communication preferably by being co-located.

**Kanban:** an ocular framework that implement Agile. It prioritize and provide continuous and small changes to the current system. It is based upon: visualize the work flow, limit work in progress, manage and enhance the flow, make policies explicit, and continuously improve.

**Scrum:** the most popular way used to implement Agile. It is an iterative model that follows a set of roles, events, and artifacts that never change. Sprints, usually take two to three weeks, allow the team to provide a deliverable software product on a regular basis. **Agile Modeling (AM):** used for documentation and modeling of software systems and it is used with other Agile methodologies like Scrum, Extreme Programming (XP). It is not considered as complete software process. It just improve models with code, but it doesn't contain programming actions.

**Rational Unified Process (RUP):** Created by IBM, it is iterative framework used for software development. The key aspects of RUP include a risk-driven process, use case focused development, and architecture-centric design.

**Scaled Agile Framework (SAFe trademark logo):** it is a very structured method to help large companies to adopt Agile. It is based on Lean and Agile principles. It has three levels: team, program, and portfolio.

**Rapid Application Development (RAD):** it puts more dependency on development rather than planning. It is an incremental model, where each component is developed in parallel. Phases of RAD are: business modeling, data modeling, process modeling, application generation, and testing and turnover.

**Empirical Control Method:** used with Agile, to help it take decisions depending on the realities observed during project. The empirical model has three components: visibility, inspection, and adoption.
2.2 Agile methods

2.2.2 Scrum development process

A framework where developers can address complex adaptive problems, while producing and delivering SW products with the highest possible quality [Sch04]. It was firstly introduced in 1990 for complex product development [SS13]. It is composed from various processes and techniques. It is Consisting of of Teams and each team has a corresponding role, artifacts, events, and rules. Each component within the Scrum has a specific purpose and is important to the project’s success. These rules link together the roles, events, and artifacts, controlling the interaction between them.

2.2.2.1 Scrum Theory

Scrum is based on the empirical process control theory [SS13]. Empiricism declares that knowledge comes from previous experiences and making decisions based on the previous knowledge. Scrum help to enhance the predictions in projects and to control risk by using an repetitive, incremental methods. The implementation of empirical process is based on three important bases: transparency, inspection, and adaptation.

**Transparency**
All sides of the process should be clear and well understandable for the people responsible for the final output. There are many factors that will lead to the required transparency like common language between participants in the project and to clearly define when a job is considered "done".

**Inspection**
Scrum participants must always check the project progress toward the sprint objective. This should be done in a way that do not tackle the work routine of each individual.

**Adaptation**
The process should be adjusted in case the process differ from the required requirements, and this should be done quickly by the responsible. There are four events used in Scrum, to assure that inspection and adaptation is fulfilled: Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective. Each of These events has a maximum duration.
2 Background

2.2.2.2 Scrum Events

Sprint is the Most important part of Scrum, it is done every two to four weeks, during which a potentially shippable product is created. A new Sprint starts immediately after the conclusion of the previous one. Sprints consists of Sprint Planning, Daily Scrums, development work, Sprint Review, and Sprint Retrospective. Each Sprint could be considered as an individual project because Sprints are used normally to complete a thing. Each Sprint has a definition of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product. Sprints enhance predictability by guaranteeing inspection and adaptation of progress into the Sprint objective [Coh10]. A Sprint can be canceled before the time of the Sprint. Only the Product Owner could cancel the Sprint. A Sprint would be canceled if the Sprint Goal becomes out-of-date. This might occur if the company changes direction or with the change of marker or technology. If a Sprint is canceled, any completed Product Backlog items are reviewed. If part of the work is potentially shippable, the Product Owner typically has to accept it. All incomplete Product Backlog Items are re-estimated and put back on the Product Backlog.

![Agile Model](image)

**Figure 2.8: Agile Model**

**Sprint Planning** Event in which the Work to be done during sprint is discussed. It is created by all the Scrum Team, and it has a maximum of eight hours. The Scrum Master should assure that the event takes place and that all participants have the same understanding level of its goal. During Scrum Planning, team should know what will be produced in the next sprint.
2.2 Agile methods

Based on the product backlog of the previous sprint, and the product owner will discuss what will be produced and prioritize items in the product backlog for this sprint. And then the development team will evaluate what can be done before the next sprint. After determining the goals of this sprint, the Development team should decide how this goal will be implemented and by the end of this meeting. The development team should explain to scrum master and to product owner how the sprint goal can be achieved and how they will organize themselves as a team to accomplish the sprint goal.

**Daily Scrum** It is a 15-minute event for the Development Team, it is done only to synchronize their work and make a plan before the next daily Scrum. Daily Scrum is done by discussing about what they have done since the last Daily Scrum, the work that will be done before the next one, and about the obstacles that they face. This Daily Scrum is done at the same time and place each day. The Scrum Master makes sure that the Development Team has made this 15 minute meeting, and that only the development team are the only people who attended this meeting but the Development Team is responsible for making this Daily Scrum.

**Sprint Review** It is a four-hour meeting or shorter, that is done at the end of each Sprint to check the Increment and to get feedback from customers. During the Sprint Review, the Scrum Team and Customers discuss about what was done in the Sprint. Based on this discussion the Product Backlog during the Sprint could be changed. The development Team will discuss with customers about good and bad things that happened during the sprint, and how they overcomes these bad things. The Product Owner explains what Product Backlog items have been already implemented and about the progress of progress regarding to the completion date. The Scrum Master makes sure that the Scrum Review takes place and that participants have understand its goals. The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint based on the discussion between Scrum team and customers, budget, market place of the product when it will be released.

**Sprint Retrospective** The Sprint Retrospective is a three-hour meeting for each Sprint, it is done after the Sprint Review and before the next Sprint Planning. It gives chance for the Scrum Team to judge themselves and to create a plan for improvements to be executed during the next Sprint. The Scrum Master make sure that this event is done and that participants understand its purpose. He also should give advices to the team members about how to be more effective, productive for the next sprint, and how to overcome difficulties faced during previous Sprint.
2 Background

2.2.2.3 Scrum Team

The Scrum Team consists of a Product Owner, Scrum Master, and the Development Team. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. The Cross-functional teams have all the skills and authorities needed to accomplish their tasks without assistance from others which are not part of the team [Sch04].

In Scrum, there are continuous feedback between the Scrum team themselves in the same company, and between the company’s Scrum team and the customers since products are always done and reviewed in a repetitive and incrementally way, which will provide better collaboration inside the company between the Team and outside with the Customers.

**The Product Owner** He is the main stakeholder responsible for increasing the value of a certain product by Optimizing the value of the work the Development Team performs. He is the only person authorized for managing the Product Backlog; he should make sure that the Product Backlog is clear by expressing Product Backlog items in a simple and clear way and that the Product Backlog is well understandable by all development team. The Product Owner is the only one authorized to tell the Development Team to adjust their work according to some new requirements and he is also authorized to change the priorities in the product backlog according to Customers desires.

**The Development Team** It consists of Software Engineers who do the work of delivering a potentially shippable product at the end of each Sprint. Each of the Development Team members have an area of interest (e.g. Software Development or Software Testing), so that all skills related to Software Engineering will be available within Development Team.

The development Team size ranges from three to nine members. Less than three will result in small productivity and more than nine will increase the complexity and will be difficult to manage. The Development Team is self dependent, the development team should do there tasks without any intervention from Other Scrum Team members, this intervention could only be during the defined Scrum Meetings.

**The Scrum Master** The Scrum Master is responsible for making sure that the Scrum Team work is compatible to normal Scrum rules. The Scrum Master directs the interactions from outside the Scrum Team to maximize the value made from the Scrum Team. He Helps the Product Owner to well understand his roles in the Scrum Process and to produce an effective Product Backlog. He removes any of the development team ambiguity regarding their roles, artifacts, meetings of the Scrum process by coaching
2.2 Agile methods

and teaching the Scrum Process clearly. He works with other Scrum Masters within Company to make the application of Scrum is done in an effective way.

2.2.2.4 Scrum Artifacts

Scrum artifact is the work or value to assure transparency and provide many opportunities for adaptation and inspection. Artifacts are designed to increase transparency of the information provided so that all scrum participants have the same level of understanding of the artifacts [Sch04].

**Product Backlog** The Product Backlog is an ordered prioritized list of things that needs to be implemented within a project, it is considered as the single source of requirements for any changes to be made.

The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering. The Product Backlog is dynamic, it continuously changes according to the change of product, Customer, and marketplace.

The refinement of the Product Backlog is done as a result of the collaboration between the Product Owner and the Development team. The Scrum Team decides how and when these refinements is done. And it can be updated at any time by the Product Owner.

The items with higher priority has more details than those with low priorities in the bottom of Product Backlog. Product Backlog items that will occupy the Development Team for the upcoming Sprint are refined so that any one item can reasonably be “Done” within the Sprint time-box. Product Backlog things that could be Done by the Development Team within one Sprint are considered Ready to be chosen in a Sprint Planning.

The Product Owner tracks the progress of any project in each sprint by comparing the work done with the work remaining to reach the project goals. And this information should be delivered to the Customers, this can be presented by using cumulative flows or burndowns for example.

**Sprint Backlog** It defines the set of tasks needed to be done in order to implement items from the Product Backlog and transform them into "Done" items. During Scrum, only the development team can modify the tasks needed to implement a certain Item of the Product Backlog during Sprint by adding or removing tasks. Elements should be removed from the Sprint Backlog when they are considered not needed.
2 Background

2.2.3 XP Practice (Test Driven Development)

Test-Driven Development is a technique for building software incrementally [Gre11]. So that, no production code is written before firstly writing a failing unit test. In TDD Tests are small and automated. Instead of starting with the production and make testing later, the TDD user expresses the desired behavior of the code in a test, the test fails, then they write the code, making the test pass.

Test automation is an important factor in TDD. In each step, a new automated unit tests are written, followed by code satisfying these automated tests. When the production code grows, the number of unit tests will also increase. With every code change, the test code runs, checking the code’s new function and also checking all the old code to assure that it is matching with the latest change in the code.

Software is fragile [Gre11]. Just about any change may result in undesired consequences. When the tests are manual, one can’t grant to run all the tests that are needed to catch unintended consequences. The retesting code is too high, so we rerun the manual tests we think are needed. Sometimes we’re not too lucky and defects are created and go undetected. In TDD, the tests help detect the unintended consequences, so when changes are made, prior behavior is not compromised.

Test-Driven Development is not considered as a testing technique [Gre11], despite a lot of valuable automated tests are written. It is just a way to solve programming problems and helps software developers make good design decisions. Tests track the production code’s required behavior.

TDD helps to make robust software without the need of long debug sessions. There is a feel of achievement and obvious progress to the goal With each test.

2.2.3.1 Test Driven Development versus Traditional Debug Later

Let’s compare TDD to the classical way of programming called Debug Later Programming (DLP) [Gre11]. In the Debug Later programming, the code is tested after being developed.

Developers usually make some mistakes during programming. But the problem with Debug-Later Programming, the discovery of those programming mistakes may take a long time (up to months). And it will be difficult for the developer to learn from those mistakes and maybe he can repeat those mistakes and those mistakes can reproduce a bunch of errors and bugs.
From the Figure 2.9 of Debug Late Programming, the time to find the defect’s main cause ($T_{find}$) is directly proportional with the time to discover a bug ($T_d$). The $T_d$ has no effect on $T_{fix}$ the most of the time. But if the mistake increase as a result of other code wrong assumption, $T_{fix}$ can increase. Some bugs could be left undetected unintentionally for many years [Gre11].

Figure 2.9: Debug Later Programming

Now from the Figure 2.10 of TDD Cycle Time. The time to find the bug ($T_{find}$) will be approximately zero, if time to discover a bug ($T_d$) is approximately zero. The code bugs is now clearer than the preceding DLP, and the programmer can get back to an early working system by removing the last edition he has done to the code.

$T_{find} + T_{fix}$ time is so small when compared to DLP, so TDD gives a quicker feedback to the programmer[Gre11].

Figure 2.10: TDD Cycle Time
2.2.3.2 Test-driven development cycle

1. Add a test
The first step in TDD is to write test, this test should be written based on a clear understanding of the requirements of the software project. This Test could also be an edition of an already existing code. This test is written in Java using the assertion command, which used to declare an expected boolean condition, and it will output false or test failed if the condition is false.

2. Run all tests and see if the new test fails
This check that all tests are working correctly, and that the new test does not pass by mistake. This step also tests the test itself. The new test should fail.

3. Write the code
In this step, code should be written so that the failed test can pass. The code written in this stage is ideal and maybe it could pass some errors without detection. But this is not a big problem, since it could be edited later in the refractoring stage.

4. Run all tests
If the tests pass, this mean that the new written code is compatible with the project’s requirements. And if the tests do not pass, the programmer has to change the code until
the test pass.

5. **Refactor code**
The Refactoring is done by removing duplications in the code. The name of Object Oriented elements like Classes, Methods, and Objects should be edited to match its purpose and objective. The cycle should then be repeated by adding another test.
2 Background

2.2.3.3 TDD Benefits

The TDD has many benefits to the Software (SW) development process [Gre11]:

- Fewer bugs:
  TDD find the small and large errors quickly, and prevent the occurrence of bad consequences as a result of logical errors.

- Small debug time:
  TDD reduce the debug time, since there are less debugs.

- Reliable documentation:
  TDD test cases are considered a better documentation than the documentation written in words.

- Fun and rewarding:
  TDD provide a new tested functionality at each time the developer code.

- Better design:
  TDD provides a testable code, and reduce the functions time and complexity which can lead to more testable codes.

- Progress tracking:
  The TDD monitor the done and remaining work.

- Peace of mind:
  TDD provides a testable, organized, and comprehensible codes. This has a good impact on the developers confidence and satisfaction.

2.2.3.4 TDD Benefits for Embedded Systems

The most faced problem by the embedded developers is the dependency of the code on the HW. Fortunately, there are a solution for this HW dependency. The embedded developer can benefit from the previous advantages of using TDD, plus some other benefits that we will discuss which are specific to embedded developers [Gre11].

- Make the developer able to test the developed code called Code Under Test (CUT) independently from the HW, if the HW is not yet delivered to the company, or in case the HW is expensive.

- Reduce the compiling time done on HW by removing bugs by using the TDD.

- Reduce the time of debug done on the HW.
2.2 Agile methods

- Help the developer to differentiate between the problems result from the interaction between the HW/SW and other problems, this is done by separating the HW from SW and replace the HW interactions with other SW alternatives.
- Make the SW design better, since the SW functionalities are separated from each other and from the HW.

2.2.3.5 Test Doubles

![Figure 2.12: Isolating dependencies](image)

Sometimes it is very difficult to test a code, because this code is dependent on other components (Third-party Library) that can not be integrated in the testing environment, so this component has to be replaced with something called Test Double. This test double is not exactly the same behavior of the replaced component but it is just a mere representation of its behavior. When the Code Under Test (CUT) interacts with the Test Double, it will not know that it is not using the real API (Third-party Library).

There are many variations of Test Double we can use, we have to remember that we do not need to implement the entire interface of the Test Double but, just the functionality needed for our CUT is needed to be provided. We can use different Test Double variations can be used for the same CUT.

There are some things that we should taken into consideration while using the Test Doubles, the first thing is that since we test the CUT under different environment from the one used in the actual production process so we have to write at least one test case that verify that the CUT work independently from the Test Double. The second thing is not to replace or edit parts of the CUT because this may result in a testing of a software with different (wrong) behavior than the original one. The third one is not overuse the Test Double which will result in an Overspecified Software.
2 Background

There are several Test Double variations (Dummy Object, Test Stub, Test Spy, Mock Object, and Fake Object) [Kos07]. They are classified according to the purpose and the cause of using the Test Double.

![Test Doubles Diagram](Figure 2.13: Types of Test Double)

**Test Stub**
It is used to replace a real component (Third-part library) that the CUT depend on. It allows the test to force the CUT to paths it might not execute. The Test Stub has a fixed responses that is used by the test code.

**Test Spy**
Is is as a modified version of a Test Stub. Like a Test Stub, a Test Spy may need to provide values to the CUT in response to method calls like the test Stubs. Test Spy are used usually as state recorder, that's mean capturing parameters passed by the CUT and the test code use the assertions to make sure that these parameters are passed correctly.

**Mock Object**
It is used to verify the behavior and identify that the CUT is with these dependencies correctly, so these Mock objects should be programmed with expectations that should be received by the CUT, it is different than the Test Spy that the assertions are made by the Mock object as expectations and not in the test code.

**Fake Object**
It is a lightweight implementation used to replace the functionality of the real API (Third-party Library), it behaves the same as the real implementation but of coarse in a simpler way. It has input and output, it gets its input through the call from the CUT and then produce an Output that will be used often in the The test code (test cases). It is commonly used when the API (Third-party Library) is not yet available, is slow, or cannot be used in the testing environment.
2.2 Agile methods

**Dummy Object**
it is considered as simple stub that is never called, its main role is to satisfy linker, compiler, or runtime calls [Gre11]. The Dummy Object have not any effect on the behavior itself and the Dummy Object can be simply used as a replacement for a null pointer.
2 Background

2.3 Safety

2.3.1 General safety standards, development process, and techniques

Nowadays, safety play an important role in embedded systems, its importance increases day by day because embedded systems are used in many applications like mobile phones, cars, medical instruments, and in plants. This continuous evolution and usage of embedded systems in many fields is proportional to the evolution reached in the field of HW and SW.

The safety analysis play nowadays an important role in the design process of safety critical systems where a defect or fault in the system will result in a dangerous situation in which the life of humans will be affected, these safety analysis is done to prevent the causes of these dangerous situations. These causes are called hazards.

Before going into details in the safety analysis techniques, we have to differentiate firstly to compare safety with reliability from one side, and safety with security on the other side. The reliability is that the system will perform its operations in a reliable and satisfactorily way. And security is the protection against dangerous conditions. So the safety should be done firstly since it is more efficient and with less cost than done later.

The functional safety is a part of safety which depend on the correct functioning and reaction of the systems components to the input of this system. And it is also the early discovery of the dangerous situations that may result in a hazard, this discovery will result in the prevention of this dangerous situation or hazard. The functional safety is an important in the automotive industry development ranging from the requirements analysis to testing phase.
2.3 Safety

2.3.1.1 IEC 61508 and ISO 26262

61508 is an international standard concerned with the functional safety in many fields like electronic, electrical and computer based electronic systems [Arm10]. It is based on two main concepts: Safety Integrity Level (SIL) and Safety Life cycle (SLC).

The SLC is the sequence of steps needed to achieve the desired functional safety. The safety life cycle is done according to the sequential flow of planning so that the system is
2 Background

complied with the safety requirements, execution, validation, and then documentation. The documentation is done to assure that the cycle meets its goals, and then changes can be done by using same sequential cycle if required functional safety is not achieved yet [exi06].

From Figure 2.14, The safety life cycle has 16 phases which could be separated into three groups:

- Phases 1-5 Analysis phase, what should we do?
- Phases 6-13 Realization phase, how we will do it?
- Phases 14-16 Operation phase, how do we keep it doing what it should?

SIL is the order of magnitude levels of risk decreasing, it is considered as an indicator for safety performance. There are four SILs defined in IEC 61508. The lowest level of risk decreasing is SIL1, and the SIL4 is the highest one.

<table>
<thead>
<tr>
<th>SIL</th>
<th>Probability of Failure on Demand ($PFD_{avg}$)</th>
<th>Safety Availability ($1-(PFD_{avg})^*$100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$\geq 10^{-5}$ to $&lt; 10^{-4}$</td>
<td>99.999 to 99.99</td>
</tr>
<tr>
<td>3</td>
<td>$\geq 10^{-4}$ to $&lt; 10^{-3}$</td>
<td>99.99 to 99.99</td>
</tr>
<tr>
<td>2</td>
<td>$\geq 10^{-3}$ to $&lt; 10^{-2}$</td>
<td>99.99 to 99.0</td>
</tr>
<tr>
<td>1</td>
<td>$\geq 10^{-2}$ to $&lt; 10^{-1}$</td>
<td>99.0 to 90.0</td>
</tr>
</tbody>
</table>

Table 2.4: IEC 61508 Safety Integrity Level

Depending on the SIL level the system need to attain, the unit testing can be used. The main reason of using the Unit testing is to assure that the SW functions is tested completely, and that the Instruction, Branch, Path coverage tests are being done. The complexity of the coverage tests is directly proportional to the increase of SIL level, that’s mean the complexity will increase with the increase of the SIL level [exi06].
ISO 26262
It is considered as the application of IEC 61508 in the automotive field industry, it is currently being used with by the majority of car manufacturers over the world [Bau]. Its main goal is to discover the hazards caused by the malfunctioning of any of the Electric/Electronic Systems within cars.

International Electrotechnical Commission (ISO) 26262 is like the IEC 61508 in that it is a safety standard based on the risk, where risks are being evaluated in a quantitative way and then safety approaches are being used to avoid these dangerous risks or failures or reducing their effects. The ISO 26262 uses the same SLC concept as the IEC 61508 which aims to completely achieve the functional safety in the automotive field during the overall development process, so it will use the concept of Automotive Safety Integrity Level (ASIL) to specify the necessary safety requirements required to attain an acceptable functional safety levels [CJL+08].

The Safety goals have to be classified according to one of four safety classes, which are called Automotive Safety Integrity Level(ASIL), with the four levels being termed ASIL A to ASIL D. ASIL D representing the highest and ASIL A the lowest [Arm10]. ASIL is specified at the start of the development process.

\[
\text{Exposure} + \text{Controllability} + \text{Severity} = \text{ASIL}
\]

\textbf{Figure 2.15:} ASIL determination formula

According to the The SAFE Consortium , in their Final proposal for extension of meta-model for hazard and environment modeling [11], the Exposure, Controllability, Severity can be classified according to some aspects.

**Exposure Classifications (E)**
is the relative expected frequency of the operational conditions in which the injury can possibly happen:

- E0 Incredibly unlikely
- E1 Very low probability (injury could happen only in rare operating conditions)
- E2 Low probability
- E3 Medium probability
2 Background

- E4 High probability (injury could happen under most operating conditions)

**Controllability Classifications**
is the relative likelihood that the driver can act to prevent the injury (C):

- C0 Controllable in general

- C1 Simply controllable

- C2 Normally controllable (most drivers could act to prevent injury)
- C3 Difficult to control or uncontrollable

**Severity Classifications (S):**

- S0 no injuries, concerned with damage that cannot be classified as related to safety

- S1 Light to moderate injuries

- S2 Severe to life-threatening (survival probable) injuries

- S3 Life-threatening (survival uncertain) to fatal injuries
2.3.1.2 FMEA, FTA, STPA

Safety analysis technique which is used to track the reliability obstacles early in the development process when it is easy to take procedures to overcome these problems and which will have a great influence on the reliability.

Failure Mode and Effects Analysis (FMEA) is used to determine the probable potential failures, and their impact on the functioning of a certain product and define actions that should be taken to reduce these failures. So that the majority of failure mode being defined, all the development team should participate in producing a list of their predicted failure modes. Of course, there may be some of the failure modes that will not be covered in the developed list [Car12].

**FMEA Types**

There are many types of FMEAs, some of them are used much more than others. The types of FMEA are:

- Design: related to components and subsystems
- System: related to system functions
- Service: related to service functions
- Software: related to software functions
- Process: related to manufacturing and assembly processes

**FMEA usage**

FMEA is used when a certain product is being designed or edited according to quality factors. When the failures of an already implemented product are being analyzed. When new strategies are used to do modifications to an existing product. It is done periodically through the life cycle of the product. FMEA has many good effects on the product being developed or on the system as whole.

- increase system reliability
- make customers more satisfied
- give a higher priority to the product defect which will have a good effect for eliminating the problems
- confirm that the cause of problems is being prevented
- make the failure mode being identified and solved early
- improve the testing and reduce related cost
2 Background

FMEA Procedures
The main steps to do FMEA safety analysis

1. Understand the product properly and know how it works. The solid understanding will make it easy to do the safety analysis.

2. Make a Block Diagram that describe and clarify the functionality of the product, this Block Diagram break down the product / system into components that are connected together, The Block Diagram will be simply clarify how the FMEA could be used with that product.

3. Complete the header presented on top to the FMEA Form worksheet: name of Product/System, Prepared By, Date, Revision, and Revision Date.

4. Use the FMEA Form worksheet to list items. The Block Diagram can be used to list these items if they are components.

5. Define Failure Modes. A failure mode is the way the product fail or to meet its required functions.

6. Describe the consequences of those failure modes.

7. Define the causes for each failure mode. A “cause” is the the reason for the failure.

8. Define Current Controls (design or process). Current Controls are the methods which prevent the cause of the failure mode from occurring.

9. Determine the likelihood of Detection. Detection is an assessment of the likelihood that the Current Controls will detect the Cause of the Failure Mode or the Failure Mode itself.

10. Review Risk Priority Numbers (RPN). The Risk Priority Number is a quantitative ranking of risk of each potential failure mode/cause, it is the mathematical product of the numerical Severity, Probability, and Detection ratings:
    \[ RPN = (Severity) \times (Probability) \times (Detection) \]

11. Determine the recommended Action/Actions. “Recommended actions” are the tasks recommended by the FMEA team to mitigate the risk which results in the failure.

12. Assign Responsibility and a Completion Date for these actions.

13. Define Actions Taken. Actions Taken are the actions that is done to mitigate the effect of risk.

14. Edit the FMEA with the change of the design.
STPA
Safety analysis techniques like FTA, FMEA, have been used for many years. Compared to these methods, STPA introduced by Leveson has a different method for hazards analysis. Systems-Theoretic Process Analysis (STPA) is based on control theory, and it was designed to fit with the analysis of complex, new systems. There are many positive feedback regarding the usage of STPA in industry till now. STPA has discovered faults that the old existing safety analysis techniques like FTA, FMEA could not found. STPa has been used in many different fields, so it is expected that it will be used more and more by the time.

There are some important Safety-Related concepts that should be explained before going into more details in STPA:

- **Accident**: unintended or intended Damage to property or injury to humans.
- **Risk**: combination of the probability of accident and its dangerous.
- **Hazard**: conditions and/or actions that results in an accident.
- **failure**: the malfunctioning of a component or system.
- **Error**: methodical fault as a result of design error e.g., SW bug.

STAMP
STPA is a hazard analysis technique based on STAMP. The Systems-Theoretic Accident Modeling and Processes (STAMP) uses the system theory in place of the old existing theory of analytic reduction and reliability, to model the systems. The system theory classify the systems as an hierarchical structure where higher levels of architecture affect the lower levels, so the system is taken as whole and not separately like the analytic reduction and reliability theory.

STAMP includes three fundamental concepts:

1. **The hierarchical control structure**: The objective of STAMP is to use the system theory in place of the traditional used theories which are based on the linear causality, the system theory model the system in hierarchical structure where the higher level control the lower ans so on. In STAMP, the system is composed of components and feedback loops.

2. **Safety constraint**: Safety Constraints are set of conditions that should not to be break so that the system do not not face dangerous consequences or hazards.
3. **the process model:**
Model used to describe the controlled process so that the control decision is done correctly, variables are used in the process model that represent the current state of a model. The process model should be used in all the levels of the hierarchy of STAMP.

**Developing a STAMP model**
STAMP model should be done before starting in the STPA hazard analysis of a certain system. This includes four parts: identifying possible accidents, identifying system hazards, identifying system constraints, and creating the system safety control structure. STAMP is a top-down fashion [Lev11]. That’s mean we should start with modeling and then proceed into details about subsystem.

1. **Identifying accidents**
Nancy Leveson define the accident as "an undesired or unplanned event that results in a loss of human life or human injury, property damage, environmental pollution, mission loss, etc." [Lev11]. Accidents should not be limited to actions that causes human damage, but they extend to include the failure to do some systems tasks. So, accidents must be divided into several categories according to their severity.

2. **Identifying hazards**
Hazard is defined as “a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident” [Lev11]. Hazards are defined on the system level, and if the hazards could not be stopped on the system level, it should be navigated to subsystem level.

3. **Identifying constraints**
Constraints are the factors that prevent the occurrence of hazards. The output of this phase will be used in the analysis of the potential failures.

4. **Creating the safety control structure**
the STAMP could be used both for existing systems and in the design of new systems. In the first case, the control structure is already known and should be modeled. In the second case, the safety control structure is implemented according to the constraints and which act as an input to the design process.

**STPA hazard analysis method**
STPA is used to discover causalities about how the system safety constraints could be breached. STPA is used in the design of a new system or for an already implemented system. It has two main steps:

1. **Identifying unsafe control actions**
Identify the potential for insufficient control of the system that could lead to a
hazardous state. Hazardous states result from inadequate control of the safety constraints, which can occur because:

- The control action required for safety is not used.
- An unsafe control action is provided.
- Control action is supplied too late or too early.
- Control action needed for the safety is stopped too early or applied for a long period of time.

In practice this step should be done by selecting a control action and creating a context table. The first column of the context table shows which control action is being analyzed. The next three columns show the process model variables which together represent the context of the process. The last three columns indicate whether the control action is hazardous or not in a given context. This way each of the control actions are systematically analyzed, and as a result unsafe control actions are identified.

2. Identifying causal factors
Determine how each unsafe control action identified in the previous step could happen.

- For each of the unsafe control action, the related parts of the control loop should be checked to see if they are the cause of this unsafe control action.
- Taking into consideration the degradation of the designed controls with time
2 Background

FTA
Fault Tree Analysis (FTA) is a safety analysis method which analyses the system from top to down, it figure out the events which could result in an accident. FTA uses a graphical symbol for simplicity and ease of understanding. There are some steps that should be followed for creating the FTA analysis.

steps of creating FTA
1. Define the events which may result in an accident.
2. Determine the causes of the unwanted event
3. Passing through all events until reaching the basic cause of these unwanted events
4. Draw fault tree diagram
5. Evaluate the FTA analysis

Benefits of FTA
There are a lot of benefits from using FTA.

• FTA is a graphical representation which clarify the relation between the undesired events and its causes of a certain system
• FTA take the whole system components into consideration, ranging from HW, SW, to human errors.
• FTA evaluate the effectiveness of redundancy in the system and decides if this redundancy is needed or not.
• FTA helps determining the main cause of many failures, thus preventing this cause may result in preventing multiple failures.
• FTA focuses on one fault at one time.

Fault Tree Diagram Symbols and Notations
There are two types of FTA notations: logic gates and events.

Events Symbols and Notations

• Basic failure event: represented as a circle.
• External event: represented with a symbol that look like a house. It is an event that will occur inescapably.
• Undeveloped event: represented with the symbol diamond. It is an event that could not be analyzed because there are no enough informations about it.
• Conditioning event: represented by an ellipse which specifies a certain condition.
• **OR gate**: output event occurs if at least one of the input events of the gate has occurred.

• **AND gate**: output event occurs if all the input events of the gate has occurred.

• **Exclusive OR gate**: output event occurs if one of the input events occurred, and not if all input events occurred.

• **Priority AND gate**: output event occurs when all input events occurred in a specific order.

• **Inhibit gate**: output event will occur if all input events occurred in addition to another conditional event.
3 Concept

3.1 ODA Control Structure

Nowadays, the ODA is a key factor for the unmanned vehicles, it is considered as the most important functionality that should be developed firstly in any unmanned vehicle or any mobile robot. The ODA has two main basic roles: obstacle avoidance and obstacle detection. The obstacle detection means that the robot detect or notice the presence or absence of a certain obstacle. The obstacle avoidance should be executed after the obstacle detection to assure that the robot will move with no collision with the obstacle. There are many methods that could be used for ODA. These methods are ODA using stationary ultrasonic sensor, rotating ultrasonic sensor, and laser sensor [MOD02]. We will start by the working principle of the ultrasonic sensor.

**Working principle of Ultrasonic Sensors**

Ultrasonic sensor is a sensor used to measure the distance to obstacles, it has been used for many years in many fields and applications like the sensors used during car parking, which are placed in the rear-end of car to detect the distance to obstacles so that the car do not collide with obstacles during parking.

The ultrasonic is composed of a transducer and drive electronics [MOD02]. The drive electronics consist of analogue and digital, the digital part is used to produce high frequencies. From Figure 3.1, the main idea of the ultrasonic distance that the transducer will produce high frequency signals called ping and then the transducer switches to the receiving mode where it waits for a specific amount of time for a return echo in the opposite direction, this echo will return in case of an obstacle collision occurred [HS11].
The distance is measured by the ultrasonic sensor according to Newton’s law, by calculating the difference between the time of ping and echo signal, and then this time will be multiplied by the velocity of signal in the air 345 m/s.

**ODA Algorithm**

The ODA developed during my master thesis is done via a rotational robot in place of stationary robot which will detect obstacles in multiple angles. Three Motors A, B, C will be used. Motor A, B will be used to control the direction of the robot and the Motor C is attached to the Ultrasonic sensor to rotate the ultrasonic sensor in multiple angles. The Ultrasonic Sensor will detect the Obstacles which is closer than 60 cm.

The code in listing 7.2 which describes the ODA algorithm will be explained by using the state diagram in figures 3.2 and the flowchart of figure 3.3, The ODA Algorithm works as follows: If the Button ODA (Button of Obstacle Avoidance Detection Robot) is being pressed, the ultrasonic Sensor will measure the Distance to Obstacle is greater than or equal than 60, the Robot will move forward. If the distance is less than 60, the Robot will move the Ultrasonic Sensor in the three angles(0, 90, 180) to detect Obstacles in these three angles, then the Robot will move in the direction of no obstacle and then the Ultrasonic Sensor will return back to initial position.
3.1 ODA Control Structure

Figure 3.2: Obstacle Avoidance Detection Algorithm State Diagram
while Stop == false

While Distance to Obstacle > 60

Move Forward

While Distance to Obstacle ≤ 60

Stop

IF First Detected Obstacle

Rotate Ultrasonic Sensor by 90° Clockwise

Rotate Ultrasonic Sensor by 90° Anti-Clockwise

IF Obstacle in the Three Directions

Robot rotate By 180°

Robot Move to The Direction with no Obstacle

Figure 3.3: Obstacle Avoidance Detection Algorithm Flowchart
3.2 Developing ODA with TDD

In this Section, we will learn how to apply the Test Driven Development to develop the Obstacle Avoidance Detection Algorithm. Before going into details, we will firstly introduce the concept of assertions used with TDD.

Assertion is a Java statement that is used to test the assumptions of the developer for a certain program, the assertion contain boolean expression which will check the correctness of the programmer's assumptions. If the output of the boolean expression is true, this is considered as a validation for the assumptions of the programmer.

The assertion can be done in two ways:

- The first is "assert e1 ", where e1 is a boolean expression. The output of this statement will be "no failure " if e1 is true, and "failure occurred " or assertion error in case the boolean expression is false.

- The second form is "assert e1 : e2 ", where e1 is a value and e2 is an expression. The output of this statement will be "no failure " if e2 has the same value as e1, and "failure occurred " or assertion error in case e2 has not the same value as e1.

As mentioned in the previous chapter, that we need to remove the dependency of the CUT on the third-party libraries (APIs). We will use the concept of test doubles, which will replace these third-party libraries (APIs). The test double variation that i have decided to use is the Fake objects will be used and which represent and emulating the real behavior of the leJOS classes.

Many test cases has been written step by step to check the functionality of the ODA robot. I have used eight test cases during TDD, these test cases were written before the ODA algorithm itself.

From listing 7.13, the two test cases "MoveWhileDistanceIsGreaterThan60" "MoveRightWhenNoObstacleOntheRight" will be described in the following two flowcharts, in Figures 3.4 and 3.5.
Test Case "MoveWhileDistanceIsGreaterThan60"

Set the distance to obstacle to 70
Distance = 70;

Forward Method within CUT

While Sonic.getDistance() > 60

Motor.A.forward(); Motor.B.forward();
mode = FORWARD; moving = true;

Break the loop
Distance = 10;

Motors A, B in the Forward direction?

Figure 3.4: Move Forward Test Case.
3.2 Developing ODA with TDD

"MoveRightWithSonicSensorMotorOnWallRight"

Set the distance to obstacle to 10

"Stop" Method within CUT

While Sonic.getDistance() ≤ 60

UltraSonicSensorMotor.rotate(-90);

Break the loop Distance 70;

"SwitchToNewLocation" Method within CUT

Rotate Ultrasonic Sensor back to initial [pointing(-90)]

Move to Right Direction;

Ultrasonic sensor return to initial angle?

Robot moved in the right direction?

Test Pass

Test Fail

Figure 3.5: Move Right Test Case.
3.3 Improved ODA Control Structure

Before explaining how the ODA has improved the safety by using the GUI, i will first briefly explain the idea and main concepts used in GUI programming with Java.

GUI is an interface which allow a user to interact directly with a certain system, it is based on the OOP concepts [BrianC2002]. The GUI could be programmed by using two APIs: Abstract Window Toolkit (AWT) or Swing libraries. I have used the Swing since it is easier, more stable than the AWT, and also because it has better portability than the AWT.

The Swing GUI is composed of components and containers. Components are the elements of the GUI like JButtons, JLabels, JText fields [BrianC2002]. Containers are those who hold the preceding components like JFrame and JPanel.

After clarifying the structure of the Swing GUI, i will explain the event handling concept. There are three important terms which are used within the event handling mechanism: Event Sources, dispatching mechanism, and Event handler. The event source is the action that could trigger/activate the event, this could be a simple button pressing by the user. The dispatching mechanism is used to queue the events coming from the Event Source and then it will be forwarded to the event handler, so that the event handler will take the required action in response to this event. The event handlers in GUI Swing are called listeners.

The last concept that should be defined before going into the improved ODA for safety, is the concurrency concept (Multi-Threading). Sometimes the GUI stop responding because it is already executing another task, so this multi-threading introduce the concept of threads, which will be used to run difference tasks in parallel and make the GUI works without stop. In multi-threading, there two types of threads: event dispatch thread, and worker threads. The event dispatch thread is the initial one used to update the GUI and it is used with light tasks. The worker threads are the threads which are used with intensive tasks.

A Graphical User Interface has been implemented using Eclipse Integrated Development Environment (IDE) and JAVA Swing library. The GUI has many buttons as shown in Figure 3.6, there are buttons that can control the direction of the Robot( Left, Right, Backward, Forward ) buttons to increase the speed of the Robot, and button for performing the Obstacle Avoidance and Detection. The GUI is supplied with labels that show the speed of the robot, and the Distance to Obstacle. The GUI and the Lego NXT are connected via Bluetooth and the control is done from PC.
3.3 Improved ODA Control Structure

From Figure 3.7, the system is now composed of five blocks: PC GUI, NXT, Ultrasonic sensor, Motor A, Motor B, Motor C. PC GUI control the Lego robot remotely via Bluetooth, it can move the robot forward, backward, right, and left. The PC GUI can also run the Lego in the ODA method if the button ODA is pressed. And it gets real-time feedback from robot about its speed and the distance to obstacle in cm.

The NXT is considered as the brain of this system, it is considered as an interface between PC GUI on one side, and Motors A, B, C plus Ultrasonic distance on the other side. It sends the command forward, backward to the Motors A, B to control their direction and send the rotating angle to Motor C to detect obstacles in multiple directions. It is also responsible for determine the distance to obstacles based on the time of pulse and echo given by the Ultrasonic sensor. the NXT also gives a real-time feedback to the PC GUI about the distance to obstacle and speed of robot.

The Motors A,B are responsible to move the robot in all directions or to stop it. Motor C is responsible to change the angle of the Ultrasonic sensor so that it can detect obstacles in three angles. The Ultrasonic sensor is responsible to send the pulse and receive the echo in case there is obstacle, and send these values to NXT which will determine the exact distance to obstacle.
After integrating the PC GUI with the ODA, the functionality will be as shown in the flowchart of Figure 3.8. The buttons (Right, Left, Backward, Forward, Stop) can interrupt the ODA mode at any time, this was done by using the variable "stop". The buttons (+,-) are used to control the speed of the robot. The problem that i have faced during design is the GUI freezing, and i have fixed this problem by the usage of Multi threading. We will discuss that in details in the next chapter.
3.3 Improved ODA Control Structure

Figure 3.8: GUI Flowchart
3.4 Software Safety Verification

Before starting in the Software Safety Verification, we need to define firstly the difference between verification and validation, it is not always easy to explain the difference between these two terms. To make so, we have to ask the following questions: Validation: Are we building the right system? and Verification: Are we building the system right? In other words, validation is checking that the system will meet the customer’s needs while verification is checking whether the system is well-engineered, without errors. Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful [MS04].

To assure that the hazardous causes of an unsafe software’s behavior cannot happen, safety verification should be done and which involves demonstrating whether the software fulfills those safety requirements and will not result in a hazardous state.

CUT here is the Obstacle Avoidance Detection Algorithm should be edited based on The test cases in the TDD code that has been edited because it was done firstly on the acceptance and validation level, some Test Cases should be added to ensure that the Software safety requirements is verified to provide evidence that the hazardous causes cannot occur (or reduce the associated risk to a low acceptable level). The Human Interaction(GUI) is also implemented so that we also could use it to overcome Hazards that could not been prevented in the code.

The Safety Constraint based on the STPA analysis done by the supervisor "The Ultrasonic Sensor should send the distance on time", has been fulfilled by editing the Obstacle Avoidance Detection algorithm as shown in Listings 7.9, 7.10, 7.11, and 7.12 in the appendix section. From Figures 3.9 and 3.10, The Code has been edited by adding an If Statement to the Stop and Forward methods to ensure that the Robot will Stop if the Ultrasonic Sensor send the Distance with a delay larger than 1.1 second. This 1.1 delay value is measured by simulating the Obstacle Avoidance Robot on the HW level to know the average time for executing the command of reading the value of the distance to obstacle.
While Distance to Obstacle > 60

True

IF Ultrasonic Sensor send the distance to obstacle in more than 1.1s

True

Stop Motors

Break loop

Move Motors Forward

False

Figure 3.9: Forward Method after STPA
While Distance to Obstacle $\leq 60$

IF Ultrasonic Sensor send the distance to obstacle in more than 1.1s

Stop Robot

Break loop

Figure 3.10: Stop Method after STPA
3.5 Final Product

So far, a GUI with many functions has been implemented. These functions are ODA, Direction and Speed feedback, Speed control, and connect and disconnect to Bluetooth. All these functionality were developed as buttons in the PC GUI. The ODA has been modified as mentioned in the previous section so that the algorithm meet the safety requirements.

The last thing that will be added to the previous functionalities is the Parking Mode or button which will be integrated in the GUI. The algorithm works as shown in the next figure, the Lego robot will continuously follow a black line. Then rotate the Ultrasonic sensor to the right direction, so that it can detect the free places to do the parking functionality. This is done by measuring the distance to obstacle, and if this distance is greater than 60, that means that there is a free place and the robot will do parking. After finishing the parking operation, robot will stop and the Ultrasonic sensor will return to initial angle and then other GUI buttons can be used to control the robot.

![Flowchart for Lego Parking](image_url)

**Figure 3.11:** Lego Parking
4 Development process

The progress of the development process has been tracked during the Master Thesis by using the Scrum board. Scrum board is a dynamic scrum board that is used to measure the progress through the duration of sprint, it gives clear and useful feedback on the project daily progress. As shown in Figure 4.1, the Scrum Board is composed of five columns: User Stories, To Do, In Progress, Verify, and Done.

![Figure 4.1: Scrum Board](image)

The first step to start with this Scrum Board is to write the so called "User Stories", These User Stories are short sentences that describes the project requirements, it is written in the format " as a <Type of User> i have to <any goal> ". The User Stories are taken from the product backlog and then broken down into the tasks required to achieve this user story or product backlog item, these tasks forms the sprint backlog and is putted in " To Do " column. If some tasks of the total of tasks in the " To Do " column will be executed, they should be moved to the " In Progress " column. Then after the " In Progress ", the code should will be reviewed, tested, and moved to the " Verify " column. After the tasks are done successfully, they should be moved to the " Done " column.

The time for each sprint was setted to be approximately one month. The sprint meeting
was done between me and the supervisor to discuss about what have been done, and what will be done in the next sprint. The daily scrum was also been done by just me, and it was as a self assessment for the work of previous day, and what i should done so that the required increment is delivered on time at the next sprint meeting.

In the next four sections, we will describe what we have done during each Sprint. We have performed the Master Thesis in four Sprints.

4.1 The first sprint

**Literature review, hardware preparation**

- Read about Lego Robot and its features, capabilities, HW limitations, Sensors provided, communication, and programming methods used with the Lego.

- Read about the Obstacle Detection Avoidance algorithms used, and then decided which will be more suitable to the Lego Robot Hardware limitations and capabilities.

- Build the Robot for the Obstacle Avoidance Detection by using Nxt brick, three Servo Motors, Ultrasonic Sensor, black rubber balloon tires, Battery Pack. These Components are linked and connected together by using Connectors, worm gear, beams, axels provided with the Lego. The Three Servo Motors are connected to the output Ports A, B, C via wire. Two Servo Motors A, B are connected to the Wheels to control the Robot direction and Servo Motor C is attached to the Ultrasonic Sensor to detect obstacles in nulli directions. The Ultrasonic Sensor is connected via wire to the input Port 4.

- Select a programming language which is cross platform (works with Linux in my case), fast, supporting the Multithreading and Bluetooth communication, which have good technical support and the availability of ready implemented projects over the internet. Then, I decided to use the Java Programming Language. To use the Java I have realised that I have to use the Lejos Firmware.

- Read about Lejos Firmware, know how to install it on the NXT Brick, and how to control the Robot Sensors, and motor by using this Firmware.

- Choose Eclipse IDE as a Development Environment. Eclipse is a development environment that is used for programming, it supports several programming languages (C, C++, Java,......) . Plugins can also be used within the Eclipse, these Plugins can be used to provide the Eclipse developers with more functionality for their code.
4.2 The second sprint

- Install the leJOS plug-in on the Eclipse.
- Upload the Lejos firmware on the NXT Brick through the Eclipse leJOS plug-in, to be able to run the Java Programming Language on the Lego NXT.
- Download the Lejos libraries (PC and Run Time). The Lejos PC library is used mainly for the Lejos programs that run on PC to remotely control the Lego Robot via Bluetooth or USB. The other run time library is used when programs are uploaded completely to the Lego robot via Bluetooth or USB, without any remote control from PC. These libraries should be included in the build path of the project.
- Read about the Test Driven Development (TDD).
- Write about the Lego and Lejos Firmware in the Master Thesis Report.

After finishing the first sprint, the Lego Robot was constructed for Obstacle Avoidance and detection as shown in Figure 4.2, and I have selected to use the Java Programming Language to implement the ODA algorithm.

![ODA Lego Robot](image)

Figure 4.2: ODA Lego Robot

### 4.2 The second sprint

**Basic functions development + safety related functions development**

- Decided to use the Java JUnit Test Framework, which is used to make unit testing in Java [Bec03].

```java
import org.junit.Test;

Listing 4.1: Java Junit Test Framework importing
```

- Read about the Assertions command in Java. Assertions are Java statement are used for testing purposes, it validates the data expected from the tester. It is a main part of the TDD test cases.
import static org.junit.Assert.assertEquals;

Listing 4.2: Assertions importing

- write the Test code, which is composed of group of Test Cases. I have started with the Test Case "MotorsMoveForwardWhenNoObstacle()", which checks that the Robot will move forward in case no obstacle in front of it. I set the the distance to obstacle to 70, and then called the method of the CUT to be tested "Forward", and then use the assertion command to assure that motors A, B will move forward by using the two command "assertEquals(true, Motor.A.isForward()); " and "assertEquals(true, Motor.B.isForward());" in the test code.

@Test
public void MotorsMoveForwardWhenNoObstacle() {
    FakeODA.stop=false;
    FakeODA.Sonic.setDistance(70);
    FakeODA.Forward();
    assertEquals(true, Motor.A.isForward());
    assertEquals(true, Motor.B.isForward());
}

Listing 4.3: MotorsMoveForwardWhenNoObstacle Test Case

- The next step is to try making the test pass by writing the method "Forward" within CUT called FakeODA, the problem was the dependency of the CUT on the Lejos third-party motor library(API) which are responsible to move the motors A and B. And we want to remove the dependency on this library so that we can test independently from the HW.

public class FakeODA {
    public static FakeUltrasonicSensor FakeSonic = new FakeUltrasonicSensor();
    public static FakeMotor MotorsOfTwoWheels = new FakeMotor();
    public static FakeMotor UltraSonicSensorMotor = new FakeMotor();
    public static int pos = -90;
    public static float select = 0;
    public static boolean FirstAngle;
    public static void Forward() {
        while (FakeSonic.getDistance() > 60 && stop=false) {
            Motor.A.forward();
            Motor.B.forward();
            FakeSonic.setDistance(5);
        }
    }
}

Listing 4.4: Forward Method in CUT

- Read about the Test Double used in the TDD, and which solve the problem of dependency of the CUT on this Third-party Library.
• Decide to use the Fake Objects and not other Test doubles variations like Test Spy, and Mock Objects.

![Diagram](image.png)

**Figure 4.3:** Lejos Motor Isolating dependencies

• As shown in the previous figure, we will break the dependency by Writing motor Fake Objects which behave the same as the original Lejos motor API. I have written two methods "forward()" and "isForward()" in the fakeMotor Class. The method "forward()" is done by setting a string variable called mode to "FORWARD" string value, and the boolean variable moving to true value. And the method "isForward()" is done to check if the motor moves forward by comparing the variable mode and the string value "FORWARD".

```java
@Test
public class BasicMotor {

    public int mode;
    // mode values
    protected static final int FORWARD = 1;

    public boolean moving;

    public void forward() {
        mode = FORWARD;
        moving = true;
    }

    public boolean isForward() {
        return mode == FORWARD;
    }
}

public class Motor {

    public static final BasicMotor A = new BasicMotor();
```
public static final BasicMotor B = new BasicMotor();
public static final BasicMotor C = new BasicMotor();

Listing 4.5: Three Motors instantiation

- The ultrasonic sensor will perform the obstacle avoidance algorithm, if the distance to obstacle is less than or equal 60. So we need to set the distance to obstacle to test the test case. So the test will continue to fail because of the absence of the Ultrasonic sensor library. So i have implemented Fake UltrasonicSensor as showed in the next figure. I have done the method "setDistance" method inside the FakeUltrasonicSensor Class, which will set the distance to obstacle that should be seen by the ultrasonic sensor.

![Diagram](image.png)

Figure 4.4: Lejos Ultraonic Sensor Isolating dependencies

public class UltrasonicSensor {
    static int Distance;

    public void setDistance(int x) {
        Distance = x;
    }
}

Listing 4.6: Fake setDistance Method within Fake Ultrasonic sensor Class

- test the functionality of the CUT forward method with the Lego robot.
4.2 The second sprint

- write the Test Case "MotorsStopWhenDetectingObstacles() " which will check that motors will stop when there is an obstacle in front of robot in a distance less than or equal 60 by Using two assert methods.

```java
@Test
public void MotorsStopWhenDetectingObstacles() {
    FakeODA.stop = false;
    FakeODA.Sonic.setDistance(10);
    FakeODA.Forward();
    FakeODA.Stop();
    assertEquals(false, Motor.A.isMoving());
    assertEquals(false, Motor.B.isMoving());
}
```

**Listing 4.7: MotorsStopWhenDetectingObstacles Test Case**

- Tried to make the test pass, by writing the method " Stop " within CUT which should stop the Motors when there is an obstacle, Then setting the ultrasonic value of distance to obstacle to 10.

```java
public static void Stop() {
    while (sonic.getDistance() <= 60 && stop = false) {
        Motor.A.stop();
        Motor.B.stop();
        FakeSonic.setDistance(70);
    }
}
```

**Listing 4.8: Stop Method within CUT**

- Adding two Methods called "isMoving()" and "stop()" in the BasicMotor class (Fake Motor). The method "isMoving()" will return a boolean true if the motor is moving and false if it is not moving. And the method "stop()" will make the boolean moving variable to false.

```java
public class BasicMotor {

    public int mode;
    // mode values
    protected static final int FORWARD = 1;
    private static final int BACKWARD = 2;
    private static final int STOP = 3;
    private static final int Right = 4;
    private static final int Left = 5;

    public boolean moving;
}
```
4 Development process

    public void stop() {
        mode = STOP;
        moving = false;
    }

    public boolean isMoving() {
        return moving;
    }
}

Listing 4.9: Fake stop and isMoving Methods within FakeMotor Class

- The test pass with no failure.
- Integrate the two functionalities of the CUT (forward, stop) method and test on the Lego robot.

    public class FakeODA {
        public static FakeUltrasonicSensor FakeSonic = new FakeUltrasonicSensor();
        public static FakeMotor MotorsOfTwoWheels = new FakeMotor();
        public static FakeMotor UltraSonicSensorMotor = new FakeMotor();

        public static void Forward() {
            while (FakeSonic.getDistance() > 60 && stop=false;) {
                Motor.A.forward();
                Motor.B.forward();
                FakeSonic.setDistance(5);
            }
        }

        public static void Stop() {
            while (sonic.getDistance() <= 60 && stop=false;) {
                Motor.A.stop();
                Motor.B.stop();
                FakeSonic.setDistance(70);
            }
        }
    }

Listing 4.10: Forward and Stop methods within CUT

- Realize Bluetooth communication between Lego and PC through pairing. The pairing was done by using a PC USB dongle and the built-in bluetooth device inside the Lego Robot, it was done from the PC by just writing the pin of the LEGO (0 0 0 0) on the PC USB dongle.
- test the functionality of the CUT stop method with the Lego robot.
• Development of a GUI by using the Swing library and implement Buttons(Right, Left, Forward, Backward, Stop) which will control the Lego Robot Direction.

• Implement Button Right which will move the robot right.

```java
btnRight = new JButton("Right");
btnRight.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.stop();
        Motor.B.forward();
        Delay.msDelay(500);
        Motor.A.forward();
    }
});
btnRight.setBounds(313, 138, 117, 25);
frame.getContentPane().add(btnRight);
```

Listing 4.11: GUI Right Button

• Implement Button Left which will move the robot left.

```java
btnLeft = new JButton("Left");
btnLeft.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.forward();
        Motor.B.stop();
        Delay.msDelay(500);
        Motor.B.forward();
    }
});
btnLeft.setBounds(52, 138, 117, 25);
frame.getContentPane().add(btnLeft);
```

Listing 4.12: GUI Left Button

• Implement Button Forward which will move the robot forward.

```java
btnForward = new JButton("Forward");
btnForward.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.forward();
        Motor.B.forward();
    }
});
```

Listing 4.13: GUI Forward Button

• Implement Button Backward Backward will move the robot backward.
btnBackward = new JButton("Backward");
btnBackward.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.backward();
        Motor.B.backward();
    }
});
btnBackward.setBounds(184, 187, 117, 25);
frame.getContentPane().add(btnBackward);

Listing 4.14: GUI Backward Button

• Implement Button Stop which will stop the robot.

btnStop = new JButton("Stop");
btnStop.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stop = true;
        Motor.A.stop();
        Motor.B.stop();
    }
});
btnStop.setBounds(184, 138, 117, 25);
frame.getContentPane().add(btnStop);

Listing 4.15: GUI Stop Button

• Development of buttons(+,-) that control the speed of the Robot.

JButton IncreaseSpeed = new JButton("+");
IncreaseSpeed.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.setSpeed((Motor.A.getSpeed()) + 100);
        Motor.B.setSpeed((Motor.B.getSpeed()) + 100);
    }
});
IncreaseSpeed.setBounds(286, 289, 70, 10);
frame.getContentPane().add(IncreaseSpeed);

Listing 4.16: GUI "+" Button

JButton DecreaseSpeed = new JButton("-");
DecreaseSpeed.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.setSpeed((Motor.A.getSpeed()) - 100);
        Motor.B.setSpeed((Motor.B.getSpeed()) - 100);
    }
});
DecreaseSpeed.setBounds(286, 311, 70, 10);
4.2 The second sprint

```java
frame.getContentPane().add(DecreaseSpeed);
```

**Listing 4.17: GUI ".-" Button**

- Test the functionality of the GUI buttons with the robot.
- Writing about TDD, and about the developed functions in the Master Thesis report.

At the End of the second sprint, we have realised two tested methods forward and stop methods by using the TDD, these methods are responsible to move and stop the robot according to the distance to obstacle provided by the ultrasonic sensor. These forward and stop algorithms were tested by using a stationary ultrasonic sensor. In addition to these tested methods for safety purposes, we have realised a GUI that can control the Lego robot direction and speed. All these functionalities have been tested on the robot.
4.3 The third sprint

Other functions development + safety related functions development

- Replace the stationary ultrasonic sensor with a rotating one, this is done by attaching the servo motor C to the ultrasonic sensor, so that the ultrasonic sensor can detect obstacles in multiple directions.

![Robot with rotating Ultrasonic Sensor](image)

**Figure 4.5:** Robot with rotating Ultrasonic Sensor

- write the Test Case "MoveRightWhenNoObstacleOntheRight()" in the test code, which will check that motors will move right if no obstacles on the right.

```java
@Test
public void MoveRightWhenNoObstacleOntheRight() {
    FakeODA.pos = -90;
    FakeODA.Sonic.setDistance(10);
    FakeODA.FirstAngle = true;
    FakeODA.stop = false;
    FakeODA.Stop();
    FakeODA.SwitchToNewLocation();
    assertEquals(true, Motor.A.isRightMoving());
    assertEquals(true, Motor.B.isRightMoving());
}
```

**Listing 4.18:** MoveRightWhenNoObstacleOntheRight Test Case

- Tried to make the test pass, by editing the "stop" method and add the method "SwitchToNewLocation()" within the CUT. The main idea of the method "SwitchToNewLocation()", is to get the ultrasonic sensor get back to the initial position after detecting an obstacle on the right side, and to do the obstacle avoidance
necessary to move in the right direction.

```java
public static void Stop() {
    while (Sonic.getDistance() <= 60 && stop == false) {
        Motor.A.stop();
        Motor.B.stop();
        if (pos == -90) {
            Motor.C.rotate(pos);
            pos = 0;
            if (FirstAngle == true) {
                Sonic.setDistance(70);
            }
        }
        if (pos != 0 && pos != 270) {
            Motor.C.rotate(90);
            FakeODA.Sonic.setDistance(70);
        }
        select = (float) pos;
        pos += 90;
        if (pos == 360)
            break;
    }
}

public static void SwitchToNewLocation() {
    switch (pos) {
        case 90: {
            Motor.C.rotate(90);
            break;
        }
// end switch

    switch ((int) select) {
        case 0: {
            Motor.A.backward();
            Motor.A.backward();
            Motor.A.MoveRight();
            Motor.B.MoveRight();
            select = 0;
            pos = -90;
            break;
        }
// end switch
```
4 Development process

Listing 4.19: Edited Stop method after rotating Ultrasonic sensor

- Adding a Method called "isRightMoving()" in the BasicMotor class (FakeMotor), which will check if the Robot will move right or not. Then this test pass with no failure.

```java
public boolean isRightMoving() {
    return mode == Right;
}
```

Listing 4.20: Fake method isRightMoving within Fake motor class

- Testing the preceding functionality of the CUT and the test will pass.
- write the Test Case "MoveLeftWhenNoObstacleOntheLeft()" which will check that motors will move left if no obstacles on the left.

```java
@Test
public void MoveLeftWhenNoObstacleOntheLeft() {
    FakeODA.pos = 180;
    FakeODA.Sonic.setDistance(10);
    FakeODA.stop = false;
    FakeODA.Stop();
    FakeODA.SwitchToNewLocation();
    assertEquals(true, Motor.A.isLeftMoving());
    assertEquals(true, Motor.A.isLeftMoving());
}
```

Listing 4.21: MoveLeftWhenNoObstacleOntheLeft Test Case

- Try to make the test passes, by editing the "stop" method and the method "SwitchToNewLocation()" within the CUT. The added code to the "SwitchToNewLocation()" method should avoid the obstacle by moving to the left direction.

```java
public static void Stop() {
    while (Sonic.getDistance() <= 60 && stop == false) {
        Motor.A.stop();
        Motor.B.stop();
        if (pos == -90) {
            Motor.C.rotate(pos);
            pos = 0;
            if (FirstAngle == true) {
                Sonic.setDistance(70);
            }
        }
    }
}
```

86
if (pos != 0 && pos != 270) {
    Motor.C.rotate(90);
    FakeODA.Sonic.setDistance(70);
} 
select = (float) pos;
pos += 90;
if (pos == 360)
    break;
}

public static void SwitchToNewLocation() {
    switch (pos) {
        case 90: {
            Motor.C.rotate(90);
            break;
        }
        case 270: {
            Motor.C.rotate(-90);
            break;
        }
        case 360: {
            Motor.C.rotate(-90);
            break;
        }
    } // end switch

    switch ((int) select) {
        case 0: {
            Motor.A.backward();
            Motor.A.backward();
            Motor.A.MoveRight();
            Motor.B.MoveRight();
            select = 0;
            pos = -90;
            break;
        }
        case 180: {
            Motor.A.backward();
            Motor.A.backward();
            Motor.A.MoveLeft();
            Motor.B.MoveLeft();
            select = 0;
        }
    }
} // end switch
pos = -90;
break;
}

//-- end switch

• Adding a Method called "isLeftMoving()" in the FakeMotor class. which checks if the Robot will move left or not. Then this test passes with no failure.

```java
public boolean isLeftMoving() {
    return mode == Left;
}
```

• Testing the preceding functionality of the CUT.

• Write the Test Case "UltraSonicSensorMotorSecondAngle() " which will check that Servo Motor C attached to the UltraSonic Sensor - which is responsible to track the obstacles in the three angles(0,90,180) Anticlockwise - will return to initial angle 90 after detecting no obstacle in the right direction and move in this direction.

```java
@Test
public void UltraSonicSensorMotorSecondAngle() {
    FakeODA.pos = 90;
    FakeODA.stop=false;
    FakeODA.Sonic.setDistance(10);
    FakeODA.Stop();
    FakeODA.SwitchToNewLocation();
    assertEquals(0, Motor.C.getAngle());
}
```

Listing 4.22: UltraSonicSensorMotorSecondAngle Test Case

• Try to make the test passes, by editing the " stop " method and the method " SwitchToNewLocation() " within the CUT, which will rotate the Ultrasonic Sensor to initial angle.

```java
public static void Stop() {
    while (Sonic.getDistance() <= 60 && stop == false) {
        if (y > 1.1) {
            Motor.A.stop();
```
Motor.B.stop();
break;
} 
Motor.A.stop();
Motor.B.stop();
if (pos == -90) {
   Motor.C.rotate(pos);
   pos = 0;
   if (FirstAngle == true) {
      Sonic.setDistance(70);
   }
}
if (pos != 0 && pos != 270) {
   Motor.C.rotate(90);
   FakeODA.Sonic.setDistance(70);
}
select = (float) pos;
pos += 90;
if (pos == 360)
   break;
}

public static void SwitchToNewLocation() {

   switch (pos) {

      case 90: {
         Motor.C.rotate(90);
         break;
      }
      case 180: {
         Motor.C.rotate(0);
         break;
      }
      case 270: {
         Motor.C.rotate(-90);
         break;
      }
      case 360: {
         Motor.C.rotate(-90);
         break;
      }
   } // end switch

   switch (((int) select) {
      case 0: {
Motor.A.backward();
Motor.A.backward();
Motor.A.MoveRight();
Motor.B.MoveRight();
select = 0;
pos = -90;
break;
}
case 90: {
    break;
}
case 180: {
    Motor.A.backward();
    Motor.A.backward();
    Motor.A.MoveLeft();
    Motor.B.MoveLeft();
    select = 0;
pos = -90;
break;
}
case 270: {
    Motor.A.setSpeed(400);
    Motor.B.setSpeed(400);
    Motor.A.backward();
    Motor.B.backward();
    Motor.A.forward();
    Motor.B.stop();
    Motor.B.forward();
    select = 0;
pos = -90;
break;
}
}  
// end switch

• Adding a Method called “getAngle()” in the BasicMotor class (FakeMotor). which check the angle of the UltraSonic Sensor, Then this test pass with no failure.

    public int angle;
    public int getAngle() {
        return angle;
write the Test Case "UltraSonicSensorMotorThirdAngle " which will check that Servo Motor C attached to the UltraSonic Sensor will return to initial angle 90 after detecting no obstacle in the left direction and move in this direction.

```java
@Test
public void UltraSonicSensorMotorThirdAngle() {
    FakeODA.pos = 180;
    FakeODA.Sonic.setDistance(10);
    FakeODA.stop=false;
    FakeODA.Stop();
    FakeODA.SwitchToNewLocation();
    assertEquals(-90, Motor.C.getAngle());
}
```

Try to make the test pass, by writing the code within CUT which will rotate the Ultrasonic Sensor to initial angle. Then this test pass with no failure.

```java
public static void Stop() {
    while (Sonic.getDistance() <= 60 && stop == false) {
        if (y > 1.1) {
            Motor.A.stop();
            Motor.B.stop();
            break;
        }
        Motor.A.stop();
        Motor.B.stop();
        if (pos == -90) {
            Motor.C.rotate(pos);
            pos = 0;
            if (FirstAngle == true) {
                Sonic.setDistance(70);
            }
        }
        if (pos != 0 && pos != 270) {
            Motor.C.rotate(90);
            FakeODA.Sonic.setDistance(70);
        }
        select = (float) pos;
        pos += 90;
        if (pos == 360)
            break;
    }
}
```
public static void SwitchToNewLocation() {
    switch (pos) {
    case 90: {
        Motor.C.rotate(90);
        break;
    }
    case 180: {
        Motor.C.rotate(0);
        break;
    }
    case 270: {
        Motor.C.rotate(-90);
        break;
    }
    case 360: {
        Motor.C.rotate(-90);
        break;
    }
    }
    // end switch

    switch ((int) select) {
    case 0: {
        Motor.A.backward();
        Motor.A.backward();
        Motor.A.MoveRight();
        Motor.B.MoveRight();
        select = 0;
        pos = -90;
        break;
    }
    case 90: {
        break;
    }
    case 180: {
        Motor.A.backward();
        Motor.A.backward();
        Motor.A.MoveLeft();
        Motor.B.MoveLeft();
        select = 0;
        pos = -90;
        break;
    }
    }
case 270: {
    Motor.A.setSpeed(400);
    Motor.B.setSpeed(400);
    Motor.A.backward();
    Motor.B.backward();
    Motor.A.forward();
    Motor.B.stop();
    Motor.B.forward();
    select = 0;
    pos = -90;
    break;
}

}// end switch

• Development of button that implement the obstacle Avoidance and Detection algorithm using a rotating ultrasonic sensor.

JButton btnOda = new JButton("ODA");
btnOda.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stop = false;
        while (NxtGui.stop==false) {
            ODA.Forward();
            ODA.Stop();
            ODA.SwitchToNewLocation();
            Delay.usDelay(50);
        }
    }
});

Listing 4.23: ODA GUI Button

• Testing the preceding functionality of the CUT.

• Get Real-time feedback about the speed and the distance to obstacle by using Text Fields called (Speed, Distance to Obstacle) and beside them. Text Fields are used to show these values.

textField = new JTextField("Distance to Obstacle(cm)");
textField.setBounds(12, 246, 200, 25);
frame.getContentPane().add(textField);
textField.setColumns(10);
Distance = new JLabel("");
Distance.setBounds(219, 251, 117, 15);
frame.getContentPane().add(Distance);
Distance.setText(sonic.getDistance());
• According to the STPA analysis results done by the Supervisor Yang Wang, there is an unsafe control action which is that the Ultrasonic sensor could send the distance to obstacle too late which will cause hazards. So I have decided to edit the CUT (ODA algorithm) and add a test case to prevent this hazardous condition. I have written the Test Case "MotorsMoveForwardWhenNoObstacleAndDistanceToObstacleIsOnTime()" which will check that Motor will move forward when there is no obstacle, and Distance to Obstacle is sent on-time to the PC.

```java
@Test
public void MotorsMoveForwardWhenNoObstacleAndDistanceToObstacleIsOnTime() {
    FakeODA.stop=false;
    FakeODA.x = 1;
    FakeODA.Sonic.setDistance(70);
    FakeODA.Forward();
    assertEquals(true, Motor.A.isForward());
    assertEquals(true, Motor.B.isForward());
}
```

• Try to make the test pass, by adding some lines of code in the Forward method within the CUT. Then this test pass with no failure.

```java
public static void Forward() {
    while (Sonic.getDistance() > 60 && stop == false) {
        if (x > 1.1) {
            Motor.A.stop();
            Motor.B.stop();
            break;
        }

        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.forward();
        Motor.B.forward();
    }
}
```
Sonic.setDistance(5);
}

Writing about the developed functions in the Master Thesis report.

After this sprint, we have edited the tested algorithm used for ODA by using a rotating ultrasonic sensor in place of the stationary one used in the previous sprint, this rotating ultrasonic sensor has been done in order to be able to detect obstacles in multiple directions.
4.4 The fourth sprint

Integration, system testing and Writing master thesis
In the last Sprint of my master thesis, Integration and System testing should be done. I will firstly define the meaning of Integration and System testing, explain the difference between them, and clarify why they are used.
System Testing, is a black box testing method which mean testing the whole system to assure that it conforms with system requirements. The System testing should be based on functional and non functional test. The non functional test is something like performance, security.
Integration testing, is a combination of white and black box testing method which tests the interaction and communication of data between the system modules and to check if there are problems occurred for the tested system after integrating the system modules.

- After Integrating the GUI with the ODA function, the system stop working, neither the ODA algorithm, nor the Distance to Obstacle provided by the Ultrasonic Sensor to the GUI label is given. So i have Used the Java GUI MultiThreading (Concurrency) concept by using the swing worker threads in place of the dispatch thread which is used initially to update the GUI. These worker threads will prevent the GUI from freezing " not responding " and the distance to obstacle will be displayed without problem.

```java
SwingWorker<Void, Integer> UltraSonicDistance = new SwingWorker<Void, Integer>() {
    @Override
    protected Void doInBackground() throws Exception {
        // Simulate doing something useful.
        for (int i = 0; i < 1000; i++) {

            // The type we pass to publish() is determined
            // by the second template parameter.
            publish(sonic.getDistance());
        }
    }

    return null;

    // Here we can return some object of whatever type
    // we specified for the first template parameter.
    // (in this case we're autoboxing 'true').
} }
```

@Override
// Can safely update the GUI from this method.
protected void process(List<Integer> chunks) {
4.4 The fourth sprint

```java
int mostRecentValue = chunks.get(chunks.size() - 1);
Distance.setText(Integer.toString(mostRecentValue));
}
}
UltraSonicDistance.execute();

Listing 4.24: GUI Distance to Obstacle worker thread

- The Motor speed values was not displayed on the GUI because of the GUI freezing. So i have Used the Java GUI MultiThreading (Concurrency) as the preceding step by using the swing worker threads. These worker threads will stop the GUI from freezing "not responding" and will show the Motors speed in a normal way.

```java
SwingWorker<Void, Integer> MotorSpeed = new SwingWorker<Void, Integer>() {
    @Override
    protected Void doInBackground() throws Exception {
        // Simulate doing something useful.
        for (int i = 0; i < 1000; i++) {
            Thread.sleep(50);

            // The type we pass to publish() is determined
            // by the second template parameter.
            publish(Motor.A.getSpeed());
        }

        return null;
    }

    // Here we can return some object of whatever type
    // we specified for the first template parameter.
    // (in this case we’re auto-boxing ‘true’).
}

@Override
// Can safely update the GUI from this method.
protected void process(List<Integer> chunks) {
    int mostRecentValue = chunks.get(chunks.size() - 1);
    Speed.setText(Integer.toString(mostRecentValue));
}
}

MotorSpeed.execute();

Listing 4.25: GUI Motor Speed worker thread
4 Development process

- Make it possible to Interrupt the execution of the ODA button from other buttons by using a variable called "stop". For example, if the ODA algorithm is running and I want the robot to move to the right.
- Implement a parking button into the GUI, which makes the robot do parking if there are free spaces.
- Do Functional testing for the whole robot functionalities.
- Integrating the parts written during previous sprints within the Master Thesis report and put them in the required format by the University.
5 Limitations

5.1 Solo Scrum is not a standard Scrum

Solo Scrum is an Agile software development process that is done with great compliance with the Agile Structure but for only one single developer. The traditional view for a Software engineer is that he is an engineer that writes code alone all the time, but this traditional view is not true because most of the time developers works together in a team.

The usage of Agile development process within a team will create a social environment of work [joh12], since there are a lot of meetings and exchange of informations between the team during the Agile process. But the question is what if there are some of developers or team members that are not social and prefer to work individually, can he/she participate in the Agile process?

There is nothing that prevent a single developer from using the Agile process [joh12], and to benefit from its practices like.

- Usage of the Scrum / Task board to organize the tasks to the project goal and to continuously track the progress of project.
- Create User Stories and Product Backlog needed to implement the project.
- create tasks to implement each particular User Story and put it as Sprint Backlog.
- Produce an increments of the product in in short sprints(range from one to four week).
- Usage of TDD.
- Benefit from the Refactoring done during TDD.
- Do a self daily review about what he has done and what he should do today, and predict the time to produce an increment of the product during the Sprint.
- Make retrospectives.
- Make Sprint Meeting (with the Supervisor).
5 Limitations

- Communicate with the customer regularly (here the Supervisor), to provide feedback and make sure they know exactly what have been done and know what should be done next.

The main drawback of solo scrum, that i have faced personally is the absence of other developers to help with the difficult things. But the Internet forums and the Supervisor can mitigate the effect of the absence of the help from other developer.

So in general, the user of the Solo Scrum should choose with practices of the Agile he can benefit from and apply to its development process.

5.2 functions are ideal

Obstacles can be detected by using different methods, e.g. Stationary Ultrasonic Sensor, rotating Ultrasonic Sensor, and laser Scanner [MOD02]. The ODA algorithm in this thesis was firstly developed by using a stationary Ultrasonic Sensor, but the problem that the Lego Robot was detecting the obstacles in just one direction. So i have edited the algorithm so that the Robot can detect the obstacles in multi direction. The Ultrasonic Sensor is more economical than laser scanner and is easier to deal with since data used is less than the laser scanner. The laser can detect the obstacles more accurately but it cannot be used with Lego NXT.

5.3 Bluetooth Communication

The distance to obstacle is provided with a small delay to the PC, because of the low speed Bluetooth. This problem has been solved by providing a larger distance to obstacle distance than the real one to the ODA algorithm to compensate for this delay. And the code was edited, so that the Robot can detect the obstacle at a reasonable distance even if the distance to obstacle is provided after a certain long delay. Bluetooth could be replaced with WIFI, but it will be difficult to use WIFI with Lego because of HW limitations.
6 Conclusion and future work

During this Master Thesis, we have analyzed the effect of using the Agile Scrum with the development process of an Obstacle Detection Avoidance algorithm for a Lego Robot NXT, by using Lejos Firmware. We have used the "Solo Scrum" and not the complete Scrum since our team members were just two members (Master Student and the Supervisor), we have adapted some of the Agile Practices like using Test Driven Development, Scrum Board, Sprints, Retrospectives, and usage of the Increment concept which provides shippable product in each Sprint.

The usage of TDD has result in many benefits like having fewer bugs, less time for debugging since tests were written before the ODA algorithm, results in a good documentation and better readability so that the code could be easily edited.

The ODA algorithm was firstly developed by using a stationary Ultrasonic Sensor, but the problem that the Lego Robot was detecting the obstacles in just one direction. So i have edited the algorithm so that the Robot can detect the obstacles in multi direction. The ODA algorithm could be improved further by using a laser sensor which is more accurate than Ultrasonic Sensor.

For safety consideration a GUI is implemented. This GUI is used to control, interrupt the ODA mode, and provide a real time feedback about the Robot speed and Distance to Obstacle, and which will allow a safe drive through Human Interaction. Moreover, the ODA algorithm has been edited to meet the safety constraints of the safety analysis technique STPA regarding the timing of ultrasonic sensor.

The Lego robot was controlled via Bluetooth Communication, this communication can be improved by using Internet(Wi-Fi) since it has higher bandwidth and higher data rate (speed). The usage of this already implemented project within an Internet Of Things (IOT) framework so that more than one Robot could be controlled remotely via Internet can be a good idea.

Another aspect that can improved for this ODA Robot is the Safety and prevent Hazards for this ODA Robot by doing SW validation according to any of the safety analysis techniques (FMEA, STPA ).
7 Appendix

// ODA.java
package guiBluetooth;

import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;
import lejos.util.Delay;

public class ODA {
    public static UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    public static int pos = -90;
    public static float select = 0;

    public static void Forward() {
        long startTime1 = System.nanoTime();
        while (sonic.getDistance() > 60 && NxtGui.stop == false) {
            if ((System.nanoTime() - startTime1) / (1000000000.0) > 1.1) {
                Motor.A.stop();
                Motor.B.stop();
                break;
            }
            Motor.A.setSpeed(400);
            Motor.B.setSpeed(400);
            Motor.A.forward();
            Motor.B.forward();
            startTime1 = System.nanoTime();
        }
    }

    public static void Stop() {
        long startTime2 = System.nanoTime();
        while (sonic.getDistance() <= 60 && NxtGui.stop == false) {
            if (((System.nanoTime() - startTime2) / (1000000000.0)) > 1.1) {
                Motor.A.stop();
                Motor.B.stop();
                break;
            }
        }
    }
}
Motor.A.stop();
Motor.B.stop();
if (pos == -90) {
  Motor.C.rotate(pos);
  Delay.nsDelay(10);
  pos = 0;
}
if (pos != 0) {
  Motor.C.rotate(45);
  Delay.nsDelay(10);
}
select = (float) pos;
pos += 45;
startTime2 = System.nanoTime();

public static void SwitchToNewLocation() {
  if (NxtGui.stop == false) {
    switch (pos) {
      case 45: {
        Motor.C.rotate(90);
        break;
      }
      case 90: {
        Motor.C.rotate(45);
        break;
      }
      case 180: {
        Motor.C.rotate(-45);
        break;
      }
      case 225: {
        Motor.C.rotate(-90);
        break;
      }
    } // end switch
    switch ((int) select) {
      case 0: {
        Motor.A.backward();
        Motor.B.backward();
        Delay.msDelay(500);
        Motor.A.stop();
        Motor.B.stop();
      }
    }
  }
}
Delay.msDelay(500);
Motor.A.forward();

select = 0;
pos = -90;
break;
}
case 45: {
    Motor.A.backward();
    Motor.B.backward();
    Delay.msDelay(500);
    Motor.A.stop();
    Motor.B.forward();
    Delay.msDelay(500);
    Motor.A.forward();
    select = 0;
pos = -90;
break;
}
case 90: {
    break;
}
case 135: {
    Motor.A.backward();
    Motor.B.backward();
    Delay.msDelay(500);
    Motor.B.stop();
    Motor.A.forward();
    Delay.msDelay(500);
    Motor.B.forward();
    select = 0;
pos = -90;
break;
}
case 180: {
    Motor.A.backward();
    Motor.B.backward();
    Delay.msDelay(500);
    Motor.B.stop();
    Motor.A.forward();
    Delay.msDelay(500);
    Motor.B.forward();
    select = 0;
pos = -90;
break;
}
} // end switch
} // end IF Statement
Listing 7.1: Obstacle Avoidance Detection Code after STPA results
/ODA.java
package guiBluetooth;

import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;
import lejos.util.Delay;

public class ODA {
    public static UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    public static int pos = -90;
    public static float select = 0;

    public static void Forward() {
        while (sonic.getDistance() > 60 && NxtGui.stop == false) {
            Motor.A.setSpeed(400);
            Motor.B.setSpeed(400);
            Motor.A.forward();
            Motor.B.forward();
        }
    }

    public static void Stop() {
        while (sonic.getDistance() <= 60 && NxtGui.stop == false) {
            Motor.A.stop();
            Motor.B.stop();
            if (pos == -90) {
                Motor.C.rotate(pos);
                Delay.nsDelay(10);
                pos = 0;
            }
            if (pos != 0) {
                Motor.C.rotate(45);
                Delay.nsDelay(10);
            }
            select = (float) pos;
            pos += 45;
        }
    }

    public static void SwitchToNewLocation() {
        if (NxtGui.stop == false) {
            ...
        }
    }
}
switch (pos) {
    case 45: {
        Motor.C.rotate(90);
        break;
    }
    case 90: {
        Motor.C.rotate(45);
        break;
    }
    case 180: {
        Motor.C.rotate(-45);
        break;
    }
    case 225: {
        Motor.C.rotate(-90);
        break;
    }
}
// end switch

switch ((int) select) {
    case 0: {
        Motor.A.backward();
        Motor.B.backward();
        Delay.msDelay(500);
        Motor.A.stop();
        Motor.B.forward();
        Delay.msDelay(500);
        Motor.A.forward();

        select = 0;
        pos = -90;
        break;
    }
    case 45: {
        Motor.A.backward();
        Motor.B.backward();
        Delay.msDelay(500);
        Motor.A.stop();
        Motor.B.forward();
        Delay.msDelay(250);
        Motor.A.forward();

        select = 0;
        pos = -90;
        break;
    }
    case 90: {
        break;
    }
}
Listing 7.2: Obstacle Avoidance Detection Code before STPA results
// TestODA.java
package nxt;

// to be able to use the assertEquals function
import static org.junit.Assert.assertEquals;

// to be able to use the @BeforeClass and @Test
import org.junit.Test;
public class TestODA {

    @Test
    public void MotorsMoveForwardWhenNoObstacle() {
        FakeODA.FakeSonic.setDistance(70);
        FakeODA.MoveWhileDistanceIsGreaterThan60();
        assertEquals(true, FakeODA.MotorsOfTwoWheels.isForward());
    }

    @Test
    public void UltraSonicSendorMotorInitialAngle() {
        FakeODA.pos=-90;
        FakeODA.FirstAngle=true;
        FakeODA.FakeSonic.setDistance(10);
        FakeODA.RotateUltrasonicSensor();
        assertEquals(-90,FakeODA.UltraSonicSensorMotor.getAngle());
    }

    @Test
    public void UltraSonicSensorMotorSecondAngle() {
        FakeODA.pos=90;
        FakeODA.FakeSonic.setDistance(10);
        FakeODA.RotateUltrasonicSensor();
        assertEquals(90,FakeODA.UltraSonicSensorMotor.getAngle());
    }

    @Test
    public void UltraSonicSensorMotorThirdAngle() {
        FakeODA.pos=180;
        FakeODA.FakeSonic.setDistance(10);
        FakeODA.RotateUltrasonicSensor();
        assertEquals(90,FakeODA.UltraSonicSensorMotor.getAngle());
    }

    @Test
    public void MoveRightWhenNoObstacleOntheRight() {
        FakeODA.pos=-90;
    }
}
FakeODA.FakeSonic.setDistance(10);
FakeODA.FirstAngle=true;
FakeODA.RotateUltrasonicSensor();
FakeODA.MoveToTheFirstLocationwithNoObstacle();
assertEquals(90,FakeODA.UltraSonicSensorMotor.getAngle());
assertEquals(true,FakeODA.MotorsOfTwoWheels.isRightMoving());

@Test
public void MoveLeftWhenNoObstacleOntheLight()
{
    FakeODA.pos=180;
    FakeODA.FakeSonic.setDistance(10);
    FakeODA.RotateUltrasonicSensor();
    FakeODA.MoveToTheFirstLocationwithNoObstacle();
    assertEquals(-90,FakeODA.UltraSonicSensorMotor.getAngle());
    assertEquals(true,FakeODA.MotorsOfTwoWheels.isLeftMoving());
}

Listing 7.3: Obstacle Avoidance Detection Test Cases before STPA results
public class BasicMotor {

    public int mode;
    // mode values
    protected static final int FORWARD = 1;
    private static final int BACKWARD = 2;
    private static final int STOP = 3;
    private static final int Right = 4;
    private static final int Left = 5;
    public int angle;
    public boolean moving;
    public int Speed;

    public void forward() {
        mode = FORWARD;
        moving = true;
    }

    public boolean isForward() {
        return mode == FORWARD;
    }

    public void backward() {
        mode = BACKWARD;
        moving = true;
    }

    public boolean isBackward() {
        return mode == BACKWARD;
    }

    public void stop() {
        mode = STOP;
        moving = false;
    }

    public boolean isMoving() {
        return moving;
    }

    public void MoveRight() {

```
mode = Right;
}

public boolean isRightMoving() {
    return mode == Right;
}

public void MoveLeft() {
    mode = Left;
}

public boolean isLeftMoving() {
    return mode == Left;
}

public void rotate(int x) {
    angle = x;
}

public int getAngle() {
    return angle;
}

public void setSpeed(int i) {
    Speed = i;
}
}

Listing 7.4: BasicMotor class(FakeMotor)
package nxt;

public class Motor {

    public static final BasicMotor A = new BasicMotor();
    public static final BasicMotor B = new BasicMotor();
    public static final BasicMotor C = new BasicMotor();

}

Listing 7.5: Motor class
// UltrasonicSensor.java
package nxt;
public class UltrasonicSensor {
    static int Distance;

    public int getDistance() {
        return Distance;
    }

    public void setDistance(int x) {
        Distance = x;
    }
}

Listing 7.6: UltrasonicSensor (Fake UltrasonicSensor class)
/FakeODA.java

package nxt;

public class FakeODA {
    public static void Forward() {
        while (Sonic.getDistance() > 60 && stop == false) {
            Motor.A.setSpeed(400);
            Motor.B.setSpeed(400);
            Motor.A.forward();
            Motor.B.forward();
            Sonic.setDistance(5);
        }
    }

    public static void Stop() {
        while (Sonic.getDistance() <= 60 && stop == false) {
            Motor.A.stop();
            Motor.B.stop();
            if (pos == -90) {
                Motor.C.rotate(pos);
                pos = 0;
                if (FirstAngle == true) {
                    Sonic.setDistance(70);
                }
            }
            if (pos != 0 && pos != 270) {
                Motor.C.rotate(90);
                FakeODA.Sonic.setDistance(70);
            }
            select = (float) pos;
            pos += 90;
            if (pos == 360)
                break;
        }
    }

    public static void SwitchToNewLocation() {
        switch (pos) {

        }
    }
}
case 90: {
    Motor.C.rotate(90);
    break;
}

case 180: {
    Motor.C.rotate(0);
    break;
}

case 270: {
    Motor.C.rotate(-90);
    break;
}

case 360: {
    Motor.C.rotate(-90);
    break;
}

} // end switch

switch ((int) select) {

    case 0: {
        Motor.A.backward();
        Motor.A.backward();
        Motor.A.MoveRight();
        Motor.B.MoveRight();
        select = 0;
        pos = -90;
        break;
    }

    case 90: {
        break;
    }

    case 180: {
        Motor.A.backward();
        Motor.A.backward();
        Motor.A.MoveLeft();
        Motor.B.MoveLeft();
        select = 0;
        pos = -90;
        break;
    }

    case 270: {
        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.backward();
        break;
    }

    case 360: {
        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.backward();
        break;
    }

    case 450: {
        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.backward();
        break;
    }

    default: {
        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.backward();
        break;
    }

}
Motor.B.backward();

Motor.A.forward();
Motor.B.stop();

Motor.B.forward();
select = 0;
pos = -90;
break;
}

} // end switch

Listing 7.7: CUT before STPA
package guiBluetooth;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingWorker;
import javax.swing.UIManager;
import lejos.nxt.Motor;
import lejos.nxt.MotorPort;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;
import lejos.pc.comm.NXTCommFactory;
import lejos.pc.comm.NXTConnector;
import lejos.util.Delay;

public class NxtGui implements KeyListener {
    NXTConnector conn = new NXTConnector();
    private JFrame frame;
    JButton btnLeft, btnRight, btnForward, btnBackward, btnStop, btnConnect;
    private JTextField textField;
    private JLabel Distance;
    private JLabel Speed;
    private JLabel position;
    static UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    private JTextField textField_1;
    public static boolean stop = false;
    private JButton btnMapping;
    private JTextField txtNearPosition;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                    NxtGui window = new NxtGui();
                    window.frame.setVisible(true);
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        });
    }
}
} catch (Exception e) {
    e.printStackTrace();
}
}
}

/**
 * Create the application.
 */
public NxtGui() {
    initialize();
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 484, 393);
    // to close the frame when pressing exit
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // so that frame can not be resized during runtime
    frame.setResizable(false);
    frame.getContentPane().setLayout(null);
    frame.setTitle("NxtControl");

    btnLeft = new JButton("Left");
    btnLeft.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Motor.A.forward();
            Motor.B.stop();
            Delay.msDelay(500);
            Motor.B.forward();
        }
    });
    btnLeft.setBounds(52, 138, 117, 25);
    frame.getContentPane().add(btnLeft);
    btnLeft.addKeyListener(this);

    btnForward = new JButton("Forward");
    btnForward.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Motor.A.forward();
            Motor.B.forward();
        }
    });
    btnForward.setBounds(52, 138, 117, 25);
    frame.getContentPane().add(btnForward);
    btnForward.addKeyListener(this);
}

120
btnForward.setBounds(184, 90, 117, 25);
frame.getContentPane().add(btnForward);
btnForward.addKeyListener(this);

btnRight = new JButton("Right");
btnRight.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.stop();
        Motor.B.forward();
        Delay.msDelay(500);
        Motor.A.forward();
    }
});
btnRight.setBounds(313, 138, 117, 25);
frame.getContentPane().add(btnRight);
btnRight.addKeyListener(this);

btnBackward = new JButton("Backward");
btnBackward.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Motor.A.backward();
        Motor.B.backward();
    }
});
btnBackward.setBounds(184, 187, 117, 25);
frame.getContentPane().add(btnBackward);
btnBackward.addKeyListener(this);

btnStop = new JButton("Stop");
btnStop.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        stop = true;
        Motor.A.stop();
        Motor.B.stop();
    }
});
btnStop.setBounds(184, 138, 117, 25);
frame.getContentPane().add(btnStop);
btnStop.addKeyListener(this);

btnConnect = new JButton("Connect");
btnConnect.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (!conn.connectTo("*", null, NXTCommFactory.BLUETOOTH)) {
            System.err.println("Connection failed");
            System.exit(1);
        }
    }
});
btnConnect.setBounds(12, 12, 117, 25);
frame.getContentPane().add(btnConnect);

textField = new JTextField("Distance to Obstacle(cm)");
textField.setBounds(12, 246, 200, 25);
frame.getContentPane().add(textField);
textField.setColumns(10);

Distance = new JLabel(
"");
Distance.setBounds(219, 251, 117, 15);
frame.getContentPane().add(Distance);

textField_1 = new JTextField("Speed");
textField_1.setBounds(95, 289, 117, 25);
frame.getContentPane().add(textField_1);
textField_1.setColumns(10);

Speed = new JLabel(
"");
Speed.setBounds(219, 299, 70, 15);
frame.getContentPane().add(Speed);

JButton btnOda = new JButton("ODA");
btnOda.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        stop = false;
        while (NxtGui.stop==false) {
            ODA.Forward();
            ODA.Stop();
            ODA.SwitchToNewLocation();
            Delay.usDelay(50);
        }
    }
});

btnOda.setBounds(333, 24, 117, 25);
frame.getContentPane().add(btnOda);

btnMapping = new JButton("Mapping");
btnMapping.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {
        int A = sonic.getDistance();
        Delay.nsDelay(10);
    }
});

btnMapping.setBounds(333, 24, 117, 25);
frame.getContentPane().add(btnMapping);
```java
System.out.print(A);
System.out.print("\n");
Motor.C.rotate(90);
Delay.nsDelay(10);
    int B = sonic.getDistance();
Delay.nsDelay(10);
System.out.print(B);
System.out.print("\n");
Motor.C.rotate(90);
Delay.nsDelay(10);
    int C = sonic.getDistance();
Delay.nsDelay(10);
System.out.print(C);
System.out.print("\n");
Motor.C.rotate(90);
Delay.nsDelay(10);
    int D = sonic.getDistance();
Delay.nsDelay(10);
System.out.print(D);
System.out.print("\n");
Motor.C.rotate(90);
Delay.nsDelay(10);
position.setText("1");
```
if (sonic.getDistance() > 40) {
    MotorPort.A.controlMotor(20, 1);
    MotorPort.B.controlMotor(20, 1);
    Delay.msDelay(100);
    MotorPort.A.controlMotor(25, 1);
    Motor.B.stop();
    Delay.msDelay(1200);
    Motor.A.stop();
    MotorPort.B.controlMotor(20, 2);
    Delay.msDelay(250);
    Motor.A.stop();
    Motor.B.stop();
    Delay.msDelay(250);
    MotorPort.A.controlMotor(40, 2);
    MotorPort.B.controlMotor(40, 2);
    Delay.msDelay(1000);
    Motor.A.stop();
    Motor.B.stop();
    Delay.msDelay(250);
    Motor.C.rotate(+90);
    break;
}
}
}
});
btnParking.setBounds(167, 12, 117, 25);
frame.getContentPane().add(btnParking);

JButton btnAcc = new JButton("ACC");
btnAcc.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < 100; i++) {
            while (sonic.getDistance() > 60) {
                Motor.A.forward();
                Motor.A.forward();
            }
            Motor.A.setSpeed(400);
            Motor.B.setSpeed(400);
        }
    }
});
btnAcc.setBounds(167, 49, 117, 25);
frame.getContentPane().add(btnAcc);
SwingWorker<Void, Integer> UltraSonicDistance = new SwingWorker<Void, Integer>() {
    @Override
    protected Void doInBackground() throws Exception {
        // Simulate doing something useful.
        for (int i = 0; i < 1000; i++) {
            // The type we pass to publish() is determined
            // by the second template parameter.
            publish(sonic.getDistance());
        }
        return null;

        // Here we can return some object of whatever type
        // we specified for the first template parameter.
        // (in this case we’re auto-boxing ‘true’).
    }

    @Override
    // Can safely update the GUI from this method.
    protected void process(List<Integer> chunks) {
        int mostRecentValue = chunks.get(chunks.size() - 1);
        Distance.setText(Integer.toString(mostRecentValue));
    }
};

UltraSonicDistance.execute();

SwingWorker<Void, Integer> MotorSpeed = new SwingWorker<Void, Integer>() {
    @Override
    protected Void doInBackground() throws Exception {
        // Simulate doing something useful.
        for (int i = 0; i < 1000; i++) {
            Thread.sleep(50);
            // The type we pass to publish() is determined
            // by the second template parameter.
            publish(Motor.A.getSpeed());
        }
        return null;

        // Here we can return some object of whatever type
        // we specified for the first template parameter.
        // (in this case we’re auto-boxing ‘true’).
    }

    @Override
    // Can safely update the GUI from this method.

protected void process(List<Integer> chunks) {
    int mostRecentValue = chunks.get(chunks.size() - 1);
    Speed.setText(Integer.toString(mostRecentValue));
}

MotorSpeed.execute();

btnConnect.addKeyListener(this);
frame.addKeyListener(this);
frame.getContentPane().addKeyListener(this);
}   // end
// of
// initialize

public void keyPressed(KeyEvent e) {
    String keyPressed = KeyEvent.getKeyText(e.getKeyCode());

    switch (keyPressed) {
    case "Up":
        btnForward.doClick();
        break;
    case "Down":
        btnBackward.doClick();
        break;
    case "Left":
        btnLeft.doClick();
        break;
    case "Right":
        btnRight.doClick();
        break;
    case "Shift":
        btnStop.doClick();
        break;
    case "Ctrl":
        btnConnect.doClick();
        break;
    default:
        break;
    }
}

Listing 7.8: GUI code
public static void Forward() {
    while (sonic.getDistance() > 60 && NxtGui.stop == false) {

        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.forward();
        Motor.B.forward();

    }
}

Listing 7.9: Forward Method Before Safety Analysis

public static void Forward() {
    long startTime1 = System.nanoTime();
    while (sonic.getDistance() > 60 && NxtGui.stop == false) {

        if (((System.nanoTime() - startTime1) / (1000000000.0) > 1.1)) {
            Motor.A.stop();
            Motor.B.stop();
            break;
        }

        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.forward();
        Motor.B.forward();
        startTime1 = System.nanoTime();

    }
}

Listing 7.10: Forward Method After Safety Analysis
Listing 7.11: Stop Method before STPA

```java
public static void Stop() {
    long startTime2 = System.nanoTime();
    while (sonic.getDistance() <= 60 && NxtGui.stop == false) {
        Motor.A.stop();
        Motor.B.stop();
        if (pos == -90) {
            Motor.C.rotate(pos);
            Delay.nsDelay(10);
            pos = 0;
        }
        if (pos != 0) {
            Motor.C.rotate(45);
            Delay.nsDelay(10);
        }
        select = (float) pos;
        pos += 45;
        startTime2 = System.nanoTime();
    }
}
```

```java
public static void Stop() {
    long startTime2 = System.nanoTime();
    while (sonic.getDistance() <= 60 && NxtGui.stop == false) {
        if (((System.nanoTime() - startTime2) / (1000000000.0)) > 1.1) {
            Motor.A.stop();
            Motor.B.stop();
            break;
        }
        Motor.A.stop();
        Motor.B.stop();
        if (pos == -90) {
            Motor.C.rotate(pos);
            Delay.nsDelay(10);
            pos = 0;
        }
        if (pos != 0) {
            Motor.C.rotate(45);
            Delay.nsDelay(10);
        }
        select = (float) pos;
        pos += 45;
        startTime2 = System.nanoTime();
    }
}
```
Listing 7.12: Stop Method After STPA
List of Abbreviations

AM  Agile Modeling. 30
ANSI  American National Standards Institute. 15
ASD  Adaptive System Development. 29
ASIL  Automotive Safety Integrity Level. 47
AWT  Abstract Window Toolkit. 64
CUT  Code Under Test. 40
DLP  Debug Later Programming. 36
DSDM  Dynamic Systems Development Method. 29
FDD  Feature-driven development. 29
FMEA  Failure Mode and Effects Analysis. 49
FTA  Fault Tree Analysis. 54
GUI  Graphical User Interface. 13
HW  Hardware. 24
IDE  Integrated Development Environment. 64
IOT  Internet Of Things. 101
IP  Internet Protocol. 25
ISO  International Electrotechnical Commission. 47
ITSDF  Industrial Truck Standards Development Foundation. 15
JVM  Java Virtual Machine. 24
LSD  Lean Software Development. 29
ODA  Obstacle Detection Avoidance. 14
OOP  Object Oriented Programming. 24
List of Abbreviations

PC  Personel Computer. 13
PID  Proportional Integral Derivative. 24
RUP  Rational Unified Process. 30
SIL  Safety Integrity Level. 45
SLC  Safety Life cycle. 45
STAMP  Systems-Theoretic Accident Modeling and Processes. 51
STPA  Systems-Theoretic Process Analysis. 51
SW  Software. 40
TCP  Transmission Control Protocol. 25
TDD  Test Driven Development. 13
XP  Extreme Programming. 29
Bibliography


jtangney. URL: http://testablelejos.sourceforge.net/ (cit. on p. 15).


Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

________________________________________

place, date, signature