Dumpsty

P5 Project
Group SW510E16
Software
Aalborg University
21st Dec 2016

AALBORG UNIVERSITY
STUDENT REPORT

Fourth semester at
Department of Computer Science
Software
Selma Lagerlöfs Vej 300
9220 Aalborg East, DK
http://www.cs.aau.dk/

Title:

Dumpsty

Project:

P5-project

Project period:

September 2016 - December 2016

Project group:

SW510E16

Participants:

Christian Dannesboe
Frederik Børsting Lund
Karrar Al-Sami
Mark Kloch Haurum
Lasse Lyngø Nielsen
Søren Lyng

Supervisor:

Bent Thomsen

Synopsis:

Pagecount: 39
Appendix range: X
Published 21-12-2016

Preface

This project has been constructed as part of the fourth semester project the by project group SW510E16 from Aalborg University, Software Engineering, from the period 2nd September to 21st December 2016 .
The project is based on the *Aalborg-model* study method, where problem and project based learning is the focus. The theme of this semester is to make an embedded system with real-time constraints. The project group chose to make a trash bin that should be able to catch the trash thrown at it.

Thank the people who should be thanked for helping making the project, here.

Signatures

Christian Dannesboe	Frederik Børsting Lund	Karrar Al-Sami
Mark Kloch Haurum	Lasse Lyngø Nielsen	Søren Lyng

Reading guide

The sources in the report are being referred to by the Harvard citation method. This includes a last name and a publication year in the report, and in the ***Bibliography*** chapter all the used sources are listed in alphabetical order.
An example of a source in the text could be: ***[Hurbain]****.*
If the source is on the left side of a dot, then that source refers only to that sentence and if the source is on the right side of a dot, then it refers to the whole section.

Figures and tables are referred to as a number. The number is determined by the chapter and the number of figure it appears as.
For example: The first figure in a chapter will have the number ***x.1****, where x is the number of the chapter. The next figure, will have the number* ***x.2****, etc.*
The listings of source code are also referred to as the tables and figures.

Source code in the report are listed as code snippets, and they're not necessarily the same as the source code, meaning that code snippets may be shorter than the actual source code or missing comments from the source code. In order to show that, the use of the following three dots are used: "...", which means that some of the source code isn't listed in the code snippet, as it may be long and irrelevant.

Throughout the report requirements are split into four colours with four different meanings. The colour blue refers to new requirements or focus requirements for that specific increment. The colour green refers to a requirement being fulfilled. Orange means that the requirement is fulfilled but has been changed. Red means that the requirement has been changed, but not yet fulfilled.

Process model

The process model is a meld of elements from both plan driven and agile development. The process is plan driven in so far as to include a clear goal of what the initial requirements of the system are. The process model is very agile in that, although there is a clear outline as to what the requirements are, these are incrementally approached, with a simple starting point becoming increasingly covering of the project vision through the subsequent increments.

The increments will be split up into four sections. Initially some of the already known requirements will be considered from previous increments and for the first increment from the analysis and hardware chapters, and these requirements will be the topic of the increment. After the initial requirement consideration, a design phase follows which details the design and problem considerations preceding the implementation. The design phase is then followed by an implementation phase which will explain the implementation process. The final phase is an evaluation of the increment and will concern whether the requirements were fulfilled, or an requirement needs to be altered, or if an requirement will be continued in the following increment. The evaluated requirements from a increment will then become the starting point of the next increment.

The process includes emphasis on a small group of individuals with tasks usually undertaken by two group members or more, which is to promote cooperation and attempt to avoid having a single individual involved in a task beyond their capabilities and to get a second opinion. This is also to ensure that information is quickly communicated and exchanged by group members. The process uses pair programming because it promotes better programming and is usually very motivating in socially engaging issues with cooperation of project members.

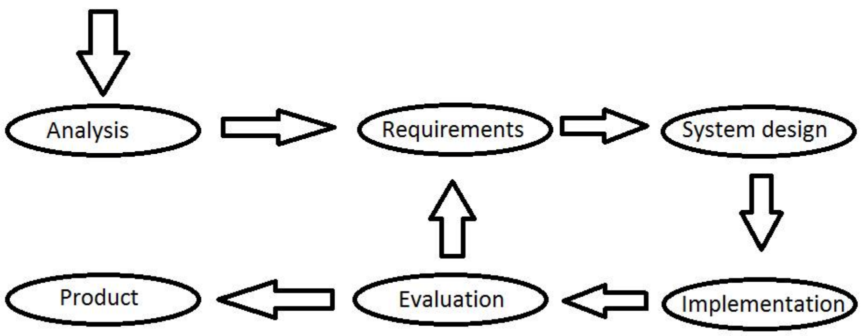

Figure 1. Process model used for the project

Contents

1 Introduction

An embedded system is a computer system which only have a few functions. It is embedded as a part of a whole device, which then also includes hardware and/or mechanical components. Embedded systems are everywhere in our everyday lives. The range for embedded systems could be all from saving lives with pacemakers to fun gadgets.[Techopedia.com]

An embedded system as a gadget, is a technological and small object or apparatus, which has a certain functionality. This functionality is limited to a certain niche.
In this project, the embedded system in consideration would be designed as a gadget, with a sole purpose of catching trash thrown at it. A trash bin that can catch trash would make the act of cleaning more fun and interactive, and the availability of each individual trash bin would be tremendously increased.

For this gadget to be usable, it must be designed as a real-time system, as this system will have certain deadlines for each task to be executed in time. The trash bin must be able to identify an object coming towards it, make some computation to identify a point to catch it from, and then compute a path to catch the object. All this must be done before the object lands on the ground, which makes the use of real-time system design a part of this project. A real-time system is a software system which is subject to a real-time constraint, and the system must control and affect an environment, by receiving data and process these, within a certain time limit.

Analysis 2

From the analysis chapter it should be possible to derive the requirements for the smart trash bin, Dumpsty. The initial user stories will help make the user requirements for the project. After the user requirements has been formed, the information from the user stories will be analysed in greater detail, depicting three different phases of the process: Throwing, detecting/tracking and catching. These three phases will be the inspiration for the system requirements. The analysis chapter will end up with a problem statement for the project.

2.1 User story

As mentioned before the user stories should help make the user requirements for the project, one user story have been made that showcase the use of Dumpsty.

Benjamin is a software engineer who is tired of wasting his precious time on the job with walking back and forth from the trash bin in his office, and therefore wants to be able to throw his trash in the general direction of the bin instead. His aim when throwing the trash isn't that of a trained basketball player, so he often has to pick up the trash after throwing it at the bin.
Benjamin wishes that the trash bin could move and collect the trash for him, so that he can throw his trash in the general direction of the bin, and the bin could then place itself in a way, that allows it to catch the trash before it lands on the floor. This would optimize the time Benjamin uses each day on collecting the trash he did not land in the trash bin.
If Benjamin throws at the bin from a designated side of a sensory camera, the robotic trash bin should identify the trash, move the bin to a place where it would be able to catch the trash, before it hits the ground.
If Benjamin throws outside a designated area of the robotic trash bin, it should not try to catch the trash, as it would compute that it is not able to get to the point of catching before the trash hits the ground.
If Benjamin and another person from his office throws trash to the trash bin at the same time, it should prioritize the first identified trash.

2.2 User requirements

The user requirements of the project have been conducted from the section 2.1. User requirements are simple requirements written in a natural language, which should help everyone get an understanding of the requirements for the project.

- The trash bin must be able to move itself to the colliding position of the trash within a certain time limit, with a certain precision.
- The trash bin should only consider trash thrown within a certain area of itself.

2.3 System capabilities

This section will define the required capabilities of the system, which will include the functionality and hardware needed to fulfill the tasks of the embedded system. This is divided into three different categories, which are the subsections of the section. These categories, along with the user requirements, will define the system requirements for the project.

2.3.1 Throwing

When considering the process of throwing the trash, certain requirements must be met, which will be simplified for this project: The trash, from here on referred to as "an object", must be round, and have a predefined and identifiable color, in our case: red.
The ball should be thrown in a high arc, to give the sensor and trash bin more time to detect, track and calculate a point, at which the robot should move to, in order to catch the object.

2.3.2 Detecting and tracking

Detecting and tracking the object are two very essential tasks, but are considered as one phase. For detecting the object, a sensor should be able to recognize the specific object. When the object has been detected, it should then be tracked by the same sensor. The sensor should track the speed of the object and the direction it is heading. This should be used to calculate the point of impact between the object and the ground.

2.3.3 Catching

The catching phase is the phase in which the trash bin calculates the impact point, and moves to that specific point. The trash bin gets information about the thrown object, in this case the impact point, from the sensor and uses the given information to calculate the impact point. What data the trash bin gets from the sensor and how the calculation is done, will be explained in later chapters.

2.4 Hardware

The following sections will describe the hardware considerations for the project, the hardware's capabilities and it's limitations. The limitations and capabilities will be in consideration when reviewing the requirements later in this chapter, to specify how the hardware can meet the requirements.

2.4.1 Sensors

Sensors were used to measure the environment around the arduino and enable the arduino to understand that environment. The sensors utilized are described in individual sections.

Microsoft Kinect

The Microsoft Kinect camera sensor enables the robot to gather information about thrown objects. The Kinect is a motion sensing device that is able to gather information about the location of an object, including the depth imaging, used to calculate the distance between the camera and the object, using a speckle pattern from an infrared camera. By using the depth information, the Kinect is able to locate the object in 3 dimensions. By spotting object multiple times in three dimensions, it is possible to calculate the path of the moving object, and thereby predict the impact point. This information can be sent to the robot, to enable the robot to move into position to possibly catch the object.

LEGO NXT Gyroscope

A gyroscope can be used to measure the heading of the robot. A gyroscope is constructed as a spinning disc that creates resistance when the robot is turned. This resistance is measured by the Lego NXT Gyro, and returned as a value representing the number of degrees per second of rotation.

LEGO NXT Accelerometer

An Accelerometer is a device that measures the force affecting it. The Lego NXT Accelerometer measures this information, and sends it to the robot, to provide capability for the robot to calculate its acceleration. The gyroscope and the accelerometer can in concoction be used to position the robot.

2.4.2 LEGO NXT Servo motor

For the robot to be able to move, it would need wheels powered by motors. In this project the LEGO NXT 9v Servo motor has been used, which at full power with no load can reach 170 RPM. The motor has a gear range of 1:48 split on the gear train in the motor. [Hurbain] The motor includes an optical fork to provide data of motor rotations down to a 1° precision.

For precision, and to benchmark the specific motors used in this project, a series of tests and measurements has been done on these motors:

The diameter of the wheel: 56mm
The circumference of the wheel: $56 \cdot \pi = 175.929$mm

The robot was programmed to drive forward for 10 seconds and was observed to travel a distance of 2550mm, which means it travels with a speed of 255mm/s.
The motor's RPM is: $(2230 \cdot 175.929) \cdot 6 = 86.966$ RPM.

2.4.3 Arduino

The Arduino Mega 2560 was chosen for this project, as it has limited computational power, which will introduce intereting problems, as well as real-time constraints. [Arduino.cc, a]

Arduino Mega 2560 Specifications:
Flash memory: 256KB (8 used by bootloader)

SRAM: 8KB
EEPROM: 4KB
Clock Speed: 16 MHz
Weight: 37g

Arduino Wifi Shield

The WifiShield was acquired to enable wifi communication between the Kinect sensor and the Arduino Mega. The intention was to send the coordinates of the objects impact point to the Arduino, without limiting the movement of the robot. The wifi connection is able to transmit data at a rate of 9600 bits a minute. [Arduino.cc, c]

Arduino Motor Shield

The Arduino Motor Shield is needed for the project to control the two DC motors independently. Without the motor shield the robot would only be able to move forward and backwards with both wheels at the same time. The motor shield needs an external power source, as the Arduino cannot provide enough power. To solve this issue, a serial circuit of two 9V batteries is attached to the shield. [Arduino.cc, b]

2.5 Problem statement & requirements

Based on the above analysis and the limitations of the project, a problem statement for the project has been constructed:

How can an embedded system control a trash bin to detect, track and catch a thrown object within a designated area?

Design specification 3

This chapter will present the requirements and the way they were conducted for the project, the requirements will be examined at the end of the report to see if they have been fulfilled. The last section in the chapter will describe the delimitations made for the project.

3.1 System requirements

The requirements for the project have been assembled through out several chapters. The Analysis chapter 2, is were the most of the requirements are conducted. The general requirements were extracted from the user story in section 2.1. Then in the Hardware section 2.4, the hardware would set some limitations for the project which lead to some more detailed requirements in the form of subrequirements.

Through out the work with the robot problems occurred and the requirements therefore changed. The changes done to the requirements though out the project, can be seen in the increments found in the Appendixes (referer til de forskellige appendixes), where the requirements for the report will be the in the Evaluation chapter (ref her) in appendix (ref til sidste increment).

The requirements for the project have been conducted thorugh out the whole process and changes were made accordingly, to adapt to the hardware's limations and the problems found. The requirements can be seen in the following paragraphs.

- The trash bin should catch the trash if the user throws it towards the trash bin and within a predefined area
  - The robots predefined area should be calculated from the hardware limitations of the motors' speed
- The robot should know where it is positioned
  - The robot should have a starting position, from where it should be able to calculate it's current position through calculations of the motory encoders
  - The robot's starting point should be placed outside its predefined area, such that it moves forward into the area
- The robot should be able to detect and track the thrown trash
  - The thrown trash should be detected and tracked by a Microsoft Kinect
  - The Kinect should send the coordinates of the impact point of the trash to the robot
- The robot should know where the the thrown trash will land
  - Trajectory prediction should be used to calculate impact point of the thrown trash

- The robot should be able to move the trash bin, such that the thrown trash lands inside the bin
  - The robot should be able to turn and drive forward
- The robot should be able to receive data from a computer, through a wireless network

3.2 Delimitations

The group have made two delimitations for the project. First the group have decided that multiple object thrown should not be considered, so only one object should be thrown at a time.

It have also been decided that the robot should not drive backwards. The robot starts outside its predefined area and will only drive forward into that. This means that if the robot is already heading towards a point in its predefined area and the Kinect sends a new point, which is behind the robot, it will not be able to drive to the new point given by the Kinect.

System design 4

4.1 Hardware choices

4.2 Software choices

4.3 Robot design

4.4 Arduino program design

As mentioned in section 4.2, the arduino program was developed in the Arduino IDE version 1.0.3. The program will be responsible to move the robot to the collision point, sent by the Kinect, of the object thrown at its predefined area. This is done by the program translating the coordinates from the Kinect, into coordinates know for the robot and move to that specific location

The program have a setup and will loop the behavior of the robot. First the sets up the WiFi connection to the Kinect program, it then waits for the Kinect program to send coordinates for the impact point of the object thrown. When it have received a set of coordinates, it will then translate the coordinates to so it knows where to move to. The next steps, the arduino program will continuously loop through until the program are exited: First it will check if it is at the impact point, if it is, the program is done. Else the robot will start moving, while it keeps adjusting it direction relative to the robots heading. Finally in the loop it will update the robots current position and its heading. All this can be broken down into tasks:

- Make connection to the Kinect
- Wait to receive coordinates from the Kinect
- When the coordinates are received, it will translate these to useful coordinates and then enter the behavioral loop function:
  - Check if already at the collision point, if it is exit program
  - Start moving, adjust direction relative to heading
  - Update the its position

4.5 Robot positioning and movement

4.6 Connecting Arduino and Kinect

For convenience, and as a part of the requirements for this project, the Arduino should receive wireless data from a computer, connected to the Kinect. The Arduino Wifi Shield makes this possible, with the Wifi library included in the Arduino IDE. The connection between the Arduino and the computer should be through a local router, with a set SSID and password. The computer should use the Kinect to calculate an impact point of the object, and send this in a specific format, so that the data can be easily read and translated to a set of coordinates for the robots movement. The computer should be able to send coordinates more than once, since the first impact point is not necessarily be precise enough to catch the object. The computer should calculate increasingly accurate impact points, which should be sent with a certain minimal inter-arrival time, to not hinder the robots movement, and yet still in time for the robot to correct itself. The Arduino should receive this data and read whenever it has sufficient time to do so, and convert the data received to match the right coordinate in the predefined area. After testing the Arduino IDE 1.6.12 with the code defined on the Arduino website's Wifi-guide [Arduino.cc, d], it was discovered that the Wifi library was not compatible in the IDE in a series of newer versions. The library was supported in version 1.0.3, which is why this particular IDE version is used in this project. The newer IDE's has increased security, which would actively refuse any TCP connection from the computer.

4.7 Scheduling

5 Increment one

In this chapter the group will go through the requirements of the project and explain how these requirements can and has been implemented. The result of this would be a new set of requirements with some of the same requirements, new requirements and changed requirements.

5.1 Requirements

The requirements considered in this increment are marked with blue colouring.

- When a user throws an object towards the trash bin, within the predefined area, the bin should always catch the object
  - The robots predefined area should be calculated from the hardware limitations of the motors' speed
- The robot should know where it is positioned
  - The robot should have a starting position, from where it should be able to calculate it's current position
  - The robot's starting point should be placed outside its predefined area, such that it moves forward into the area
- The robot should be able to detect and track the thrown object
  - The thrown object should be detected and tracked by a Microsoft Kinect
  - The Kinect should send the coordinates of the impact point of the object to the robot
- The robot should be able to calculate the impact point for the object
  - Trajectory prediction should be used to calculate the impact point of the thrown object
- The robot should be able to move the trash bin, such that the thrown object lands inside the bin
  - The robot should be able to turn and drive forward and backwards
  - The robot should be able to recognize the coordinates sent from the Kinect
- The robot should be able to receive data from a computer, through a wireless network

These requirements are rudimentary, and the very essence of the project lies within the fulfilment of these requirements. In this increment the fundamentals of the robots movement should be implemented, the predefined area should be defined, a starting point for this area should be determined and the Microsoft Kinect should be able to detect and track an object by using trajectory prediction.

5.2 System design

The following sections describe how the marked requirements in 5.1 should be fulfilled. The theories and ideas will be explained subsequently.

5.2.1 Predefined area

The robots predefined area, is an area where the robot should catch the object within. This area is being made because of the limitations of the robots motors and is based on motor speed and average time of a throw, so the predefined area is where the robot should be expected to catch the object within.

5.2.2 Throwing

The reason why the throw of the object (which in this project will be a table tennis ball) is important, is because the robot should have as much time to move to the collision point as possible. Before the robot moves to the collision point, the Kinect should calculate the where the robot should move, send the data to the robot, and the robot then moves. The Kinect can at any point send new data to the robot, so its course have to be changed, therefore it is important for the robot to have sufficient time to move within the predefined area.

5.2.3 Microsoft Kinect

To detect the object, the Kinect has to use one of its cameras, more specifically the depth sensor. The depth sensor uses a range of infrared speckles that draw a pattern in the room. Every cluster of speckles can be identified from each other, which makes the Kinect able to differentiate between objects in the room. The depth of the specific object can be determined by the use of two cameras, as both cameras can identify the speckles at the object and then triangulate the distance between the cameras and the object. This speckle pattern technology will only work indoor, and limits the use to only one kinect, as the speckles can be washed out by other lightsources. [Wikipedia.com, 2016]

5.2.4 Trajectory prediction

When the object, referred to in this section as the projectile, is detected and the tracking of that projectile has started, the trajectory can be predicted. This prediction is limited to the amount of data sent by the sensory camera, meaning that for every camera reading, one detection of the object is gained. The precision of the prediction will increase according to the time a projectile has been tracked.
In this project the outdoor weather conditions, that might affect the projectile trajectory, are not considered, since the prediction is done indoor. As well, the effects of air resistance, also called drag, will not be considered.

The trajectory of a projectile is the path of a thrown projectile affected by gravity, without propulsion. To calculate a trajectory of a projectile, the initial height, the angle which the projectile is launced from, the speed of the projectile at launch and the gravitational acceleration must be taken into account.
The initial height in this project is the height at which the projectile is detected. The

angle and speed at which the projectile is launched, will be calculated from the first few trackings after detecting the projectile. The gravitational acceleration is considered as $9.81m/s^2$, which is the common value near the earths surface.

To catch the projectile, the distance and time of flight have to be calculated. This is done with two mathematical formulas:
$g:\ the\ gravitational\ acceleration\ (9.81m/s^2)$
$\theta:\ the\ angle\ at\ launch v:\ the\ speed\ at\ launch$
$y_0:\ the\ initial\ height$
$d:\ the\ total\ horizontal\ distance\ traveled$
$t:\ the\ time\ of\ flight$
$v_{vert}:\ the\ vertical\ velocity$
$v_{hori}:\ the\ horizontal\ velocity$
$t_h:\ the\ time\ since\ first\ detection$
$d_t:\ the\ distance\ at\ time\ t$

The above variables can express the formulas needed to provide the total distance traveled by the projectile, and the time of flight.

Distance traveled:

$$d = \frac{v\ cos\ \theta}{g}(v\ sin\ \theta + \sqrt{(v\ sin\ \theta)^2\ +\ 2gy_0})$$

Time of flight:

$$t = \frac{d}{v\ cos\ \theta} = \frac{v\ sin\ \theta\ +\ \sqrt{(v\ sin\ \theta)^2\ +\ 2gy_0}}{g}$$

To predict the trajectory of the projectile, the altitude and distance of the projectile at any time during the flight must be calculated, according to the initial height (y_0):

$$y = v_{vert}t - \frac{1}{2}gt^2$$

$$d_t = v_{hori}t$$

After this section, the projectile can be tracked, and the path for a given projectile can be calculated to a certain degree of precision.

5.2.5 Gyroscope and Accelerometer

For positioning the robot, the Lego NXT Gyro and Accelerometer sensors will be considered. These sensors will be used together to position the robot, as the gyroscope will tell the number of degrees the robot has turned, which is relative to the heading of the robot at the first recording of data. This heading will be used together with an accelerometer, which can be used to calculate the distance the robot has traveled since the first recording of data. The speed, travel time and heading will be the data of these sensors.

5.2.6 Movement

The robot is expected to be able to drive both forward and backwards. The robot is expected to turn using either one active wheel, or using both wheels in a counterclockwise motion in order to turn faster.

5.3 Implementation

This section explains how the marked requirements has been implemented and whether or not the group changed the requirement, as something didn't work out as expected.

5.3.1 Gyroscope and Accelerometer in use

For the implementation of the gyroscope and the accelerometer, a test has to be done, to benchmark the precision of the sensors. For this, the data sent from the Arduino when the sensor has been plugged into the Arduino, was plotted in the Serial Plotter, which is a feature in the Arduino IDE.

The accelerometer would not be precise enough to calculate the distance the robot had moved. The accelerometer would give an acceleration, to calculate from an acceleration to a distance, the time had to be multiplied to the acceleration twice. If the acceleration was not precise, this would lead to a big margin of error.

The gyroscope worked just fine, but when it was not moving, the data seen in the Serial plotter would wander. This meant that when reading data from the gyroscope, either the gyroscope should be reset before reading data or the value would be the difference of when it started reading till it stopped.

Instead of using the gyroscope and the accelerometer, it was decided to use the motor' encoderes, which can be used to calculate the heading and the distance moved.

5.3.2 Predefined area

The robots predefined area will be calculated using the time of flight and the speed of the motors. This predefined area is limited to be strictly in front of the robot. A bouncing throw would have a travel time of 2 - 2.15 seconds, explained in greater detail in section 5.3.3. The predefined area was constructed using the robot itself. The robot was placed marking its starting point, and was set to turn a specific number of degrees and move forward for two seconds. After a series of runs, the area marked by the robot turned out as shown in figure 5.1. The predefined area measures 2851.5cm^2.

5.3.3 Throwing

To ensure that the robot is provided the maximum amount of time for calculations and movement, the average time of flight of an object at a designated distance was measured.

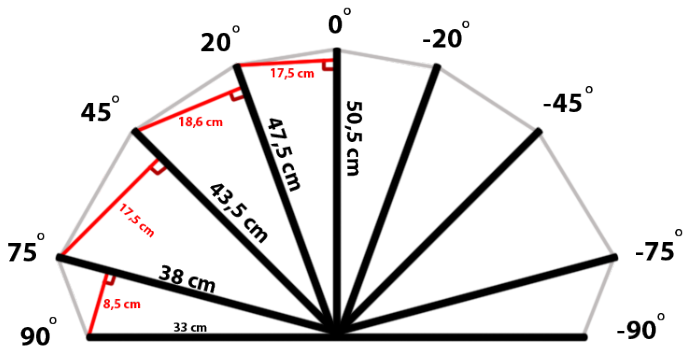

Figure 5.1. The robots predefined area

At a distance of three meters, the average time of flight of the object was 1.1 seconds, and at a distance of five meters, the average time of flight was 1.25 seconds.

With the results from the abovementioned measurements, it is concluded that a regular throw would not provide enough time for the robot to catch the object. As mentioned in the 2.4.2 section, the robots speed was calculated to be 87 RPM. With a 1.1 seconds time of flight, with a distance of three meters, the impact point should be within a 280 mm radius from the robot. This simply is not fast enough to catch the object from a three or five meter distance.

This introduces the idea of bouncing the object, in order to solve the above mentioned problem. The idea is, that the initial height of the object is considered to be the floor instead of the top half of the arc, which in theory should provide more time for the robot. After conducting another set of tests, this theory proved to be correct. The average time of flight at a distance of three meters is 2 seconds, and at a distance of five meters the average time of flight is on average 2.15 seconds. The bouncing throw provides the robot with close to double the amount of time.

5.3.4 Microsoft Kinect

5.3.5 Movement

An Arduino Motor Shield is used to enable greater control of the movement of the motors through the motor shields library called Adafruit which is included through #include <AFMotor.h> (unsure how to present this nicely) The Arduino Motor Shield and library enables manipulation of the individual wheels instead of only being able to either rotate forward or backwards on both wheels simultaneously. The wheels were tested and forward motion was achieved with a speed of roughly 25.5 centimetres per second. It was discovered that the speed at which the robot drives backwards is much slower than its forward moving speed, because of the speed of backwards motion, it was decided to not consider driving backwards at all since the speed were inadequate and since this would also simplify the problem, and could later be solved with different motors. To solve this issue, it was decided to change the throw to always be in front of the machine, thus ensuring it would never be forced to drive backwards. This does however not take into account if the Kinect projectile prediction predicts the projectile will land behind the machine but this issue only arises if prediction is sufficiently miscalculated which is unlikely.

5.4 Evaluation

This section will include a short summary of the chapter, ending up with a new requirement list, based on the how the group implemented the marked requirements from the beginning of the chapter.

The requirements stating that the robot should know where it is positioned, will persist through this increment, as no solution was found in this increment. The use of a gyroscope and accelerometer was not the right solution for this project, as the need of precise data would not be supported by sensors with a large amount of jitter. Another way of positioning the robot would be by calculating the distance traveled on each wheel, through the motor encoder in the NXT servo motors. The distance traveled would be calculated by the rotation of the wheel and the circumference of that wheel. The robots starting position will always determine where the predefined area of the robot is along with its hardware limitations. The concept of a predefined area is entirely based on the rotation speed of each motor, the wheel circumference and the duration of the throw. If any element that defines the predefined area changes, then so does the predefined area. Movement was not nearly as simple as expected and the implications of the problems encountered in its implementation is not fully solved in this increment.

The requirement of driving forward and backwards is redefined as only consisting of driving fowards. The reason for the change was explained in the section 5.2.6.

The requirements after finalizing increment one is:

- The trash bin should catch the trash if the user throws it towards the trash bin and within a predefined area
  - The robots predefined area should be calculated from the hardware limitations of the motors' speed
- The robot should know where it is positioned
  - The robot should have a starting position, from where it should be able to calculate it's current position through calculations of the motor encoders
  - The robot's starting point should be placed outside its predefined area, such that it moves forward into the area
- The robot should be able to detect and track the thrown trash
  - The thrown trash should be detected and tracked by a Microsoft Kinect
  - The Kinect should send the coordinates of the impact point of the trash to the robot
- The robot should know where the the thrown trash will land
  - Trajectory prediction should be used to calculate impact point of the thrown trash

- The robot should be able to move the trash bin, such that the thrown trash lands inside the bin
  - The robot should be able to turn and drive forward
  - The robot should be able to recognize the coordinates sent from the Kinect
- The robot should be able to receive data from a computer, through a wireless network

Increment two 6

This chapter is just like the chapter above. The results from increment one will be discussed and implemented and this should result in the same, new or changed requirements as well.

6.1 Requirements

The requirement marked with blue will be considered throughout this chapter.

- The trash bin should catch the trash if the user throws it towards the trash bin and within a predefined area
  - The robots predefined area should be calculated from the hardware limitations of the motors' speed
- The robot should know where it is positioned
  - The robot should have a starting position, from where it should be able to calculate it's current position through calculations of the motor encoders
  - The robot's starting point should be placed outside its predefined area, such that it moves forward into the area
- The robot should be able to detect and track the thrown trash
  - The thrown trash should be detected and tracked by a Microsoft Kinect
  - The Kinect should send the coordinates of the impact point of the trash to the robot
- The robot should know where the the thrown trash will land
  - Trajectory prediction should be used to calculate impact point of the thrown trash
- The robot should be able to move the trash bin, such that the thrown trash lands inside the bin
  - The robot should be able to turn and drive forward
  - The robot should be able to recognize the coordinates sent from the Kinect
- The robot should be able to receive data from a computer, through a wireless network

6.2 System design

In this section, the marked requirements should be fulfilled and will be described and explained.

6.2.1 Wifi Shield

When the Kinect have found the coordinates for the collision point of the thrown object, these coordinates should be sent to the robot, for it to be able to move to the collision point and catch the object. The data should be sent, as earlier mentioned in section 2.4.3, by the Arduino wifi shield. The Arduino and the computer running the Kinect program, should be connected to their own network, with the computer sending data to the wifi shields IP-address.

6.2.2 Robot positioning

In the previous chapter, it was decided to use the motor's encoders to calculate the heading of the robot and the distance it has moved. The encoders will keep track of the number of degrees the specific motor have turned, which can be calculated to a distance using the wheels circumference. The coordinate set for the collision point of the object and the robots current position, will be used to calculate the heading, which the robot should have to reach the collision point.

6.3 Implementation

This section explains how the marked requirements were actually implemented in the project and whether or not the requirements have been changed, removed or been fulfilled.

6.3.1 Wifi Shield

To implement the Wifi Shield as part of the Arduino, the router connecting the Wifi Shield with a computer had to be port forwarded. The private network hosted by the D-link router had a specific ssid and password, which had to be explicitly assigned in the Arduino code for the Wifi Shield. The setup in the Arduino code was used to start the connection between the computer and the Wifi Shield, as well as printing the log of the connection. The setup for the wifi shield can be found in listing 6.1.

Listing 6.1. Connecting the Wifi shield to the network

```
void setup() {
  Serial.begin(9600);

  Serial.println("Attempting to connect to WPA network...");
  status = WiFi.begin(ssid, pass);

  if ( status != WL_CONNECTED) {
    Serial.println("Couldn't get a wifi connection");
    while(true);
  }
  else {
    server.begin();
    Serial.println("Connected to network");
  }
  ip = WiFi.localIP();
  Serial.println(ip);
}
```

After connecting the Wifi Shield to the router, the Arduino is ready to receive data. The data will be sent from a laptop, which connects and sends data as shown in 6.2. A tcpClient is created and connected to the IP of the Wifi Shield, and the port that has been forwarded. The data sent in the code is a string "test", which is sent as a byte-array, and after sending the data, the tcpClient closes the connection.

Listing 6.2. Connecting the computer to the Wifi Shield

```
tcpClient.Connect("192.168.0.100", 9999);
string data = "test";
const int IntSize = 4;
byte[] bytedata;
Stream stream = tcpClient.GetStream();

stream.Write(Encoding.UTF8.GetBytes(data.ToCharArray()), 0, data.Length);
tcpClient.Close();
```

When both the computer and the Wifi shield has made the connection, and the computer has sent the data, the Arduino can then receive the data. This is done through a Wifi client, from the Arduino Wifi library. When assigning the client to the server.available, the value of the client in an if-statement would be true if any client from the server has data available for reading, or false if no data was available. If a client is connected, and the client has data available for reading, the next byte will be appended on the string readString, and after all the data available has been read, the full string sent by the client will be printed.

Listing 6.3. Receiving data from the computer

```
client = server.available();
while (client.connected()){
  if (client.available()) {
    char c = client.read();
    readString += c;
  }
}
if(readString != ""){
  Serial.println(readString);
  readString = "";
}
```

6.3.2 Robot positioning

In the top of the Arduino program, a lot of variables were declared, which can be seen in listing 6.4. The two volatile integers leftTotal and rightTotal are used to count the number of degrees the motor have turned. leftTotal and rightTotal are volatile integers because the value of the variables can be changed by other things than the code, which in this case is the motors.

motorLeftRun and motorRighRun are booleans used to check whether the motors are running or not.

The variable called heading have the value of the robot's heading, which in this project will be the degrees the robot's direction is pointing away from the y-axis of its predefined area. Then robot's current position will be saved as coordinates in the variables posX and posY.

Several integers are declared which will get further explanation later in the section. Also in the top of the program some identifiers have been defined and the necessary libraries have been included to the program.

Listing 6.4. The defined and declared variables

```
volatile int leftTotal = 0;
volatile int rightTotal = 0;
bool motorLeftRun = false;
bool motorRightRun = false;
double heading = 0.0, posX = 0.0, posY = 0.0;
unsigned int tid;
int leftTemp, rightTemp, goalX = 230, goalY = 330, margin = 10, loopcount = 0, signalCount = 0;
```

An arduino program is split up into a setup function and a loop function, where the code in the loop function should describe the robot's behaviour. The setup function are run once for every power up or reset of the Arduino board. The setup function can be found in listing 6.5. Two pins are setup as inputs for the arduino board and an interrupt is attached to each of them. Every time a change occurs for the input pins, which in this case is every time the motors have turned 1 degree, the functions incrementLeft and incrementRight are called accordingly. The two functions increments the counters leftTotal and rightTotal which are counting the number of degrees the motors have turned.
Next up the motors are turned on. This is done by setting their speed, which here is set to the highest speed possible, and then both motors are ret to RELEASE, which will stop the motors. Also the booleans motorLeftRun and motorRightRun are set to false.
At the bottom of the setup funtion, then leftTotal's and the rightTotal's values have been assigned to the temporary variables leftTemp and rightTemp.

Listing 6.5. The setup function

```
pinMode(LEFTENCODERPIN, INPUT);
attachInterrupt(LEFTPINTOINTERRUPT, incrementLeft, CHANGE);
pinMode(RIGHTENCODERPIN, INPUT);
attachInterrupt(RIGHTPINTOINTERRUPT, incrementRight, CHANGE);

motorLeft.setSpeed(255);
motorRight.setSpeed(255);

motorLeft.run(RELEASE);
motorLeftRun = false;
motorRight.run(RELEASE);
motorRightRun = false;

leftTemp = leftTotal;
rightTemp = rightTotal;
```

When the arduino program enters the loop function, seen in listing 6.6, the first thing it will do is to check the if-statement at line. This if-statement checks if the robot is at its desired position, which is the coordinate set sent from the Kinect(goalX and goalY). If the robot was at the position it will release both motors and exit the program, else it will skip this if statement and continue in the loop function.
In lines 9-12 the difference of the desired coordinate and the current coordinate of the robot are calculated and assigned to the variables deltaY and deltaX. With these new variables the heading of which the robot should have to hit the desired point. The delta heading are calculated using the goalHeading and the current heading of the robot.
In lines 14 - 23 the if-else statement checks if the robot is on the right heading. If the robot's heading is less than -0.1 it will release the right motor and drive forward with the left, making the robot turn right. The robot will do the same check for 0.1 and will release the left motor and drive forward on the right, making the robot turn left. If the robot is

on its right heading, it will drive forward, towards it desired point.
In lines 25-35 two new integers, currentLeft and currentRight, are assigned the values of leftTotal and rightTotal. Then the distance of how much the wheels have moved since last time it was looped through is calculated. This is done by taking the difference of currentLeft and leftTemp times DISTPRDEGREE, which was defined at the beginning of the program, the same is done with currentRight and tempRight. Then leftTemp and rightTemp are assigned the values of currentLeft and currentRight.
The distance the robot have moved is the average of how much the two motors have moved, so the the distances of the two motors are added together and divided by two. With the distance, the robots new current position is calculated assigned to the values posX and posY, and the new heading for the robot.

Listing 6.6. The loop function

```
1  void loop() {
2    if(round(posX) <= goalX + margin && round(posX) >= goalX - margin && round(posY) <= goalY +
         margin && round(posY) >= goalY - margin){
3      motorLeft.run(RELEASE);
4      motorRight.run(RELEASE);
5      delay(50);
6      exit(0);
7    }
8
9    double deltaX = goalX - posX;
10   double deltaY = goalY - posY;
11   double goalHeading = atan(deltaX/deltaY);
12   double deltaHeading = goalHeading - heading;
13
14   if(deltaHeading < -0.1){
15     motorLeft.run(FORWARD);
16     motorRight.run(RELEASE);
17   }else if(deltaHeading > 0.1){
18     motorLeft.run(RELEASE);
19     motorRight.run(FORWARD);
20   }else{
21     motorLeft.run(FORWARD);
22     motorRight.run(FORWARD);
23   }
24
25   delay(10);
26   int currentLeft = leftTotal;
27   int currentRight = rightTotal;
28   double deltaLeft = (currentLeft - leftTemp) * DISTPRDEGREE;
29   double deltaRight = (currentRight - rightTemp) * DISTPRDEGREE;
30   leftTemp = currentLeft;
31   rightTemp = currentRight;
32   double dist = (deltaLeft + deltaRight) / 2.0;
33   posX += (dist * sin(heading));
34   posY += (dist * cos(heading));
35   heading += (atan((deltaRight - deltaLeft) / WHEELDIST));
36 }
```

6.4 Evaluation

This section will include a short summary of the chapter and conclude with an updated requirement list, showing changes from the first requirement list in the chapter.

7 Increment three

7.1 Connecting Arduino and Kinect

Implementation 8

Tests 9

Discussion 10

Conclusion 11

12 Future work

Bibliography

Arduino.cc, **a**. Arduino.cc. *Arduino Mega 2560.* `https://www.arduino.cc/en/Main/ArduinoBoardMega2560`. Accessed: 25-09-2016.

Arduino.cc, **b**. Arduino.cc. *Arduino Motor Shield.* `https://www.arduino.cc/en/Main/ArduinoMotorShieldR3`. Accessed: 18-10-2016.

Arduino.cc, **c**. Arduino.cc. *Arduino WiFi Shield.* `https://www.arduino.cc/en/Main/ArduinoWiFiShield`. Accessed: 17-10-2016.

Arduino.cc, **d**. Arduino.cc. *WifiGuide.* `https://www.arduino.cc/en/Guide/ArduinoWiFiShield`.

Hurbain. Philippe "Philo" Hurbain. *LEGO® 9V Technic Motors compared characteristics.* `http://www.philohome.com/motors/motorcomp.htm`. Accessed: 12-10-2016.

Techopedia.com. Techopedia.com. *Embedded Systems.* `https://www.techopedia.com/definition/3636/embedded-system`. Accessed: 25-09-2016.

Wikipedia.com, **2016**. Wikipedia.com. *Kinect Wiki.* `https://en.wikipedia.org/wiki/Kinect`, 2016. Accessed: 19-10-2016.

Appendix A