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: 29
Appendix range: X
Published 21-12-2016

Preface

This project has been developed as part of the fourth semester project 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 was to make an embedded system. 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.

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(or iteratively) approached, with a simple starting point becoming increasingly covering of the project vision.

The increments will be split up into four sections. Initially some of the already known requirements will be considered, and these requirements will be the topic of the increment. After the initial requirement consideration, what is needed to fulfil the requirements will be explained, followed by an explanation of the implementation, and a final evaluation of the increment will be described. The evaluation will concern whether the requirements were fulfilled, and changes to the requirement list can happen.

The process includes emphasis on a small group with tasks usually undertaken by two group members or more, trying to avoid a single individual stuck in a task without aid from other group members. This is also to promote that knowledge gained 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 the aid of the other project members.

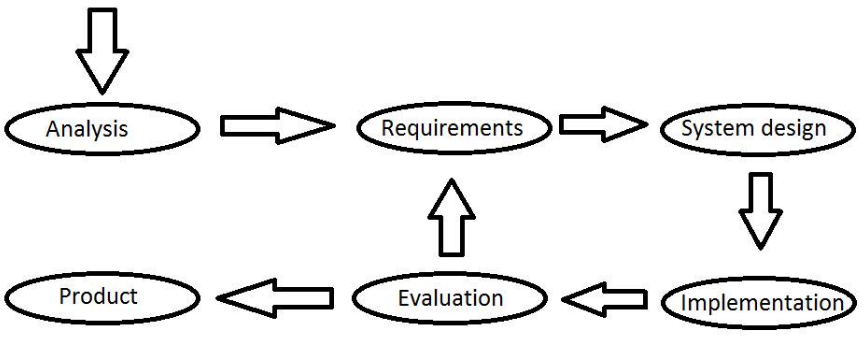

Figure 1. Process model used for the project

Contents

1 Introduction **1**

2 Analysis **3**
2.1 User story . . . 3
2.2 User requirements . . . 3
2.3 System capabilities . . . 4
2.3.1 Throwing . . . 4
2.3.2 Detecting and tracking . . . 4
2.3.3 Catching . . . 4
2.4 Problem statement & requirements . . . 4

3 Hardware **7**
3.1 Sensors . . . 7
3.1.1 Microsoft Kinect . . . 7
3.1.2 LEGO NXT Gyroscope . . . 7
3.1.3 LEGO NXT Accelerometer . . . 7
3.2 LEGO NXT Servo motor . . . 8
3.3 Arduino . . . 8
3.3.1 Arduino Wifi Shield . . . 8
3.3.2 Arduino Motor Shield . . . 8
3.4 Requirements . . . 9

4 First increment **11**
4.1 Requirements . . . 11
4.2 System design . . . 11
4.2.1 Predefined area . . . 12
4.2.2 Throwing . . . 12
4.2.3 Microsoft Kinect . . . 12
4.2.4 Trajectory prediction . . . 12
4.2.5 Gyroscope and Accelerometer . . . 13
4.2.6 Movement . . . 14
4.3 Implementation . . . 14
4.3.1 Gyroscope and Accelerometer in use . . . 14
4.3.2 Predefined area . . . 14
4.3.3 Throwing . . . 15
4.3.4 Microsoft Kinect . . . 15
4.3.5 Movement . . . 15
4.4 Evaluation . . . 15

5 Increment two **17**
5.1 Requirements . . . 17

5.2 System design 17
5.2.1 Wifi Shield 17
5.3 Implementation 17
5.3.1 Wifi Shield 17
5.4 Evaluation 18

6 Tests **19**

7 Discussion **21**

8 Conclusion **23**

9 Future work **25**

Bibliography **27**

A Appendix **29**

Introduction 1

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 waste bin, Dumpsty. The user stories will be initial and will help make the user requirements for the project. After the user requirements 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 waste 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 litter after throwing it at the bin.
Benjamin wishes that the waste 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 waste bin.
If Benjamin throws at the bin from a designated side of a sensory camera, the robotic waste 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 perimeter of the robotic waste bin, it should not try to catch the litter, 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 waste bin at the same time, it should prioritize the first identified object.

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 perimeter 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

To make it easier for the sensor to detect and track the thrown object. The object should be round of shape and in a clear colour such as red. This will help the sensor because it the object will stand out from other objects.
The ball should be thrown as slow as possible, to give the sensor and waste bin more time to detect, track and calculated the collision point for the object and the waste bin. This could be done by throwing an underhand throw or bouncing the ball at the area of the waste bin.

2.3.2 Detecting and tracking

Detecting and tracking are a very essential part and is considered as one phase. For detecting the sensor should be able to recognize the specified object within it's field of view. The waste bins area and the area in front of the waste bin should be monitored by the sensor.
When the object have been detected, it should then be tracked by the sensor. The sensor should track the speed of the object and the direction it is heading in. This should be used to calculate the objects collision with the waste bins area.

2.3.3 Catching

The catching phase of the project is for the waste bin to calculate the objects collision within the waste bins area, and the movement to the designated collision point. The waste bin gets information about the flying object from the sensor and uses the given information to calculate the collision point. What data the waste bin gets from the sensor and how the calculation is done, will be explained in later chapters.

2.4 Problem statement & requirements

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

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

The requirements for in this chapter is the requirements for the project, if there was no hardware or time constraints:

- The trash bin should catch the trash if the user throws it towards the trash bin and within a predefined area.
- The robot should know where it is positioned
- The robot should be able to detect and track the thrown thrash
- The robot should know where the the thrown thrash will land
- The robot should be able to move the trash bin, such that the thrown thrash lands inside the bin.

Hardware 3

This chapter 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.

3.1 Sensors

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

3.1.1 Microsoft Kinect

The Microsoft Kinect sensor, enables the robot to gather information about flying objects that it can catch. The Kinect is a motion sensing device that is able to gather information about the location of an object, including the depth of an object (how far it is from the sensor), using an infrared camera. By using the depth information, the Kinect is able to semi-accurately locate the object in 3 dimensions. By spotting objects multiple positions in these three dimensions, it is possible to calculate the path of the moving object, and thereby predict the position in which it is going to land. This information can be send to the robot, so the robot can move into position to possibly catch the object before it hits the ground.

3.1.2 LEGO NXT Gyroscope

A gyroscope can be used to measure rotation of the robot. A gyroscope works, by having 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.

3.1.3 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 and possibly its location.

3.2 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 motors has been used, which at full power with no load can reach 170 RPM. These motors has a gear range of 1:48, split on the gear train in the motor. [Hurbain] This motor includes an optical fork to provide the rotation sensor functionality, which can provide data of motor rotations down to a 1° precision.

The group decided to do their own calculations for the RPM and a calculation for mm/s for the servo motor.

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.

3.3 Arduino

We chose the arduino mega 2560 because we wanted to introduce real time problems and have limited computational power to introduce interesting problems as a learning experience. At the start of the project, we tried a NXT but it would've been too simple to work with. We actually ordered an arduino uno for even less memory, but there weren't any left, so we went with the mega 2560. [Arduino.cc, a]

Some specs:
Flash memory: 256KB (8 used by bootloader)
SRAM: 8KB
EEPROM: 4KB
Clock Speed: 16 MHz
Weight: 37g

For the making the robot connect to the program for the Microsoft Kinect and controlling the DC motors, an Arduino Wifi Shield and Arduino Motor shield will be used in the project and explained below.

3.3.1 Arduino Wifi Shield

The WifiShield was acquired to enable wifi communication between the Kinect sensor and the arduino mega. Intended use was for the Kinect to send the coordinates of the object or the coordinates of the object when it lands and or bounces to the arduino. The wifi connection is able to transmit data at a rate of 9600 bits a minute, which is equivalent to 1.2 bytes per millisecond. [Arduino.cc, c]

3.3.2 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 run forward and backwards with both tires at the same time, not making it possible to turn. To run the motor shield an external power source is needed, in this case two 9V batteries in a serial circuit. [Arduino.cc, b]

3.4 Requirements

Considering all the limitations and the capabilities of the hardware for the project, the requirements from the previous chapter will be reviewed and new requirements will be added if necessary. New requirements will be mark with red.

- 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
  - 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, drive forward and drive backwards
  - Multiple trash thrown should not be considered

First increment 4

Test om: Bounce, predefined area Skal komme væk fra accelerotmeter og gyroscope.

4.1 Requirements

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

- 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
  - 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, drive forward and drive backwards
  - Multiple trash thrown should not be considered

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 robot's movement should be implemented, the predefined area should be calculated, 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.

4.2 System design

The following sections describe how the marked requirements in 4.1 will be attempted to be fulfilled. The theories and ideas behind will be explained subsequently.

4.2.1 Predefined area

The robots predefined area, is an area where the robot should catch the ball 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 ball within.

4.2.2 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.

4.2.3 Microsoft Kinect

To detect the thrown object, the Kinect has to use one of it's cameras, more specifically the depth sensor in this case. The depth sensor uses a range of infrared speckles, which 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, it can 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]

4.2.4 Trajectory prediction

When the trash, 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, since the prediction is done indoor, the outdoor weather conditions that might affect the projectile trajectory is not considered. 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 without propulsion, affected by gravity. For calculating 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, and 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 $9.81 m/s^2$, which is the standard near the earths surface.
In this project we are interested in catching the projectile, and to do that, we need to

calculate the distance the projectile travels before hitting the ground, and the amount of time before the projectile hits the ground. This is done with two mathematical formulas:

$g:$ *the gravitational acceleration* $(9.81m/s^2)$
$\theta:$ *the angle at launchv* $:$ *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*

From these variables we can express the formulas needed to provide the total distance traveled by the projectile, and the amount of time this would take.
The distance traveled is:

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

The time of flight is:

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

To calculate a 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 correctness.

4.2.5 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 start of the 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 beginning of the recording of data. The time spent and the speed of the robot will be the data of this sensor.

4.2.6 Movement

The robot was expected to be able to drive forward and backwards, to enable it to at most turn 90 degrees from any given angle, since any movement exceeding 90 degrees would be performed by driving backwards and turning from the opposite end of the robot. The robot was expected to be able to turn whether using either one active wheel or make both wheels go counterclockwise each other to turn more swiftly.

4.3 Implementation

4.3.1 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. After various tests, the decision was to find a better alternative to these sensors, as the gyroscope in particular, had a lot of jitter. This jitter would even increase over time, making the use of the robot limited, as any user would have to restart the robot when the use wants to throw anything at the robot.

4.3.2 Predefined area

The robots predefined area will be calculated using the travel time of the ball and the speed of the motors used for the project. The predefined area was limited to be strictly in front of the robot as mentioned in movement. As mentioned in section 4.3.3, the bouncing throw would have a travel time of 2 - 2.15 seconds, before landing in the predefined area. The predefined area was calculated using the robot itself. The robot was placed marking its starting point, which will always be its starting point. It was set to turn a specific number of degrees and move forward, after two seconds the robot would stop. The place the robot stops is marked aswell, which lead to the predefined area seen in figure 4.1. The predefined area have an area of of 2851.5cm^2 which is 0.285m^2.

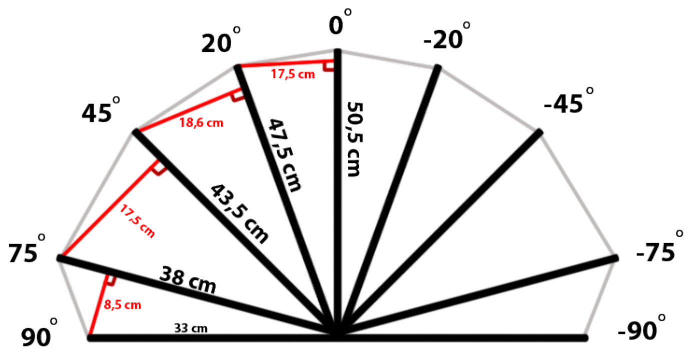

Figure 4.1. The robots predefined area

4.3.3 Throwing

Tests have been performed to ensure that the robot is provided the maximum amount of time for calculations and catching the trash. In order to do so, multiple test were conducted, and the results were acquired from a distance of 3 meter, the time it took for the trash (in this case, a table tennis ball) was in average 1.1 second. Multiple tests from a distance of 5 meter were also conducted and that provided an average of 1.25 second.

With the results from the throwing tests, the group could conclude that a normal throw wouldn't provide enough time for the robot to catch the ball, else the predefined area for the robot had to be very small, due to the limitations of the motors. As mentioned in the 3.2 section, the group calculated the robot's speed to be 87 RPM, which simply isn't enough to catch trash from a 3 or 5 meter distance. The robot would be able to catch trash within 280mm of itself, given the 1.1 second run time of a 3 meter throw, and therefore nowhere near sufficient enough.

The group decided after several tests to let the trash (still a table tennis ball) bounce on the floor before it had to be catched, which would give the robot extra time to get to the collision point of the ball. This provided nearly double the amount of time for the throw to be catched, providing an average of 2 seconds from a 3 meter throw and 2.15 seconds from a 5 meter throw.

4.3.4 Microsoft Kinect

4.3.5 Movement

Movement: The wheels were tested and forward motion was achieved with a speed of [insert speed] centimetres per second. The robot was also able to drive backwards at a lower speed of ..... Insert reason why we didn't turn by moving both wheels counterclockwise each other. 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 is inadequate (prefer tangible example) and since this would also simplify the problem. 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 (anything else we can say about this? or rephrase).

4.4 Evaluation

The requirements stating that the robot should know where it is positioned, will persist through this increment, as no solution has been found. 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 sensor 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, as the predefined area is defined by the robots position, and it's hardware limitations. The concept of predefined area is entirely based on the rotation speed of motor with wheel circumference and the duration of the throw, if any element the predefined area consists of 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 increment one.

When considering the the requirement: "The robot should be able to turn, drive forward and drive backwards", the group decided to remove the drive backwards part of the requirement as the group has decided to have a fixed starting point, from where the trash bin will always start and therefore didn't see a reason for the ability to move backwards. The other two parts of the requirement are fulfilled.

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 wheel rotations
  - 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
  - Multiple trash thrown should not be considered

Increment two 5

5.1 Requirements

5.2 System design

5.2.1 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 ot be able to move to the collision point and catch the object. The data should be sent, as earlier mentioned in section 3.3.1, by the Arduino wifi shiled. The Arduino and the computer running the Kinect program, should be connected to their own network, with the computer sending the data to the wifi shields IP-address.

5.3 Implementation

5.3.1 Wifi Shield

First a private network for the computer running the Kinect program and the Arduino wifi shield were sat up. This was done so that it was possible to port forward the necessary ports on the router, so that the Kinect program could send the data to the wifi shield.

Listing 5.1. 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);
}
```

Next thing is too connect the wifi shield to the router, which can be seen in listing 5.1. Two char arrays was declared in the top of the program, containing the network name(ssid) and the password(pass).
To connect the wifi shield to the router, the serial port is opened, to search for the network.

The wifi shield will try to find and connect to the network matching name of the ssid. If it couldn't find any networks matching the ssid is will print out "Couldn't get a wifi connection". If the connecting was made it will print out "Connected to network" and print out the ip for the wifi shiled and it is now ready to receive date from the computer.

(The code describing how it receives data is needed here!)

5.4 Evaluation

Tests 6

Discussion 7

Conclusion 8

Future work 9

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.

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