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: 31
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.

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.

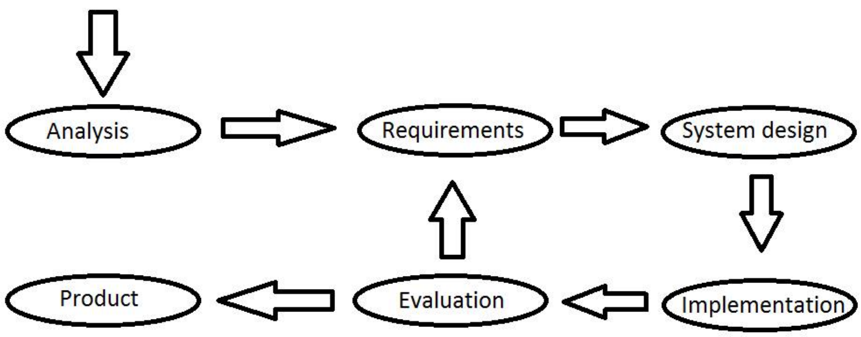

Figure 1. Process model used for the project

Contents

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 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 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?

The requirements drawn from this chapter, would be the requirements for the project, if the hardware used in this project would be the cause of any limitations:

- When a user throws an object towards the trash bin, within the predefined area, the bin should always catch the object.
- The robot should know where it is positioned.
- The robot should be able to detect and track the thrown object.
- The robot should be able to calculate the impact point for the object.
- The robot should be able to move the trash bin, such that the thrown object 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 around the arduino and enable the arduino to understand that environment. The sensors utilized are described in individual sections.

3.1.1 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.

3.1.2 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.

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. The gyroscope and the accelerometer can in concoction be used to position the robot.

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 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.

3.3 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

3.3.1 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]

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 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]

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. The new requirements will be mark with red.

- 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 receive data from a computer, through a wireless network

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.

- 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 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.

4.2 System design

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

4.2.1 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.

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 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]

4.2.4 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 $9.81 m/s^2$, 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* $(9.81 m/s^2)$
$\theta :$ *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*

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.

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 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.

4.2.6 Movement

Since any forward movement exceeding 90 degrees would be inefficient compared to moving backwards, 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.

4.3 Implementation

4.3.1 Gyroscope and Accelerometer in use

A series of tests of the gyroscope and accelerometer benchmark the precision of the sensors. For this, the data sent from the Arduino by the sensor, 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 user wants to throw anything at the robot.

4.3.2 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 4.3.3. 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 4.1. The predefined area measures 2851.5cm^2.

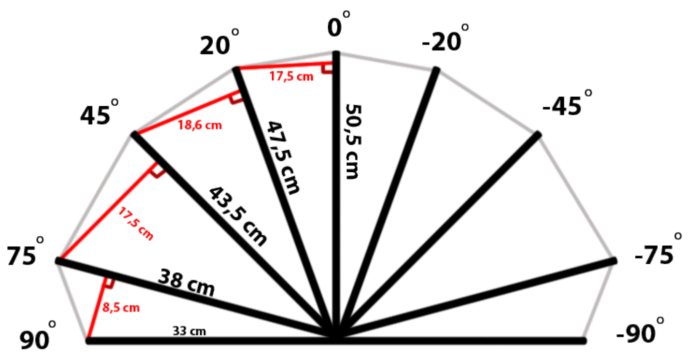

Figure 4.1. The robots predefined area

4.3.3 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.

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 3.2 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.

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 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 4.2.6.

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 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

Increment two 5

5.1 Requirements

- 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

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

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 5.1.

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);
}
```

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 send data as shown in 5.2. 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 5.2. 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 the 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 5.3. 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 = "";
}
```

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

5.4 Evaluation

Tests 6

Discussion 7

Conclusion 8

9 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.

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