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

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

Synopsis:

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.

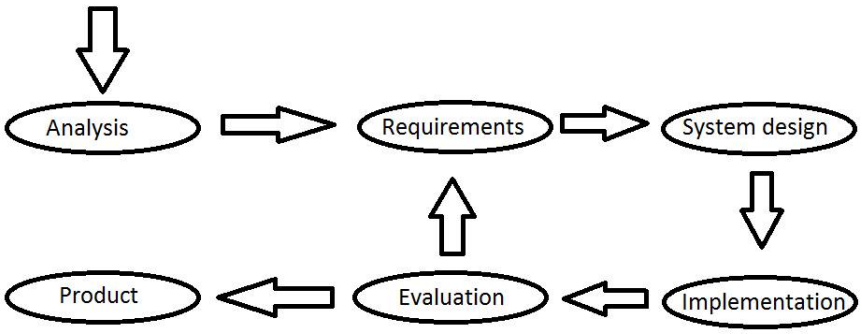

Figure 1. Process model used for the project

Contents

1 Introduction

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.

Theory 3

3.1 Field of view

Through out the report there will be mentioned the field of view and the predefined area. The field of view is the area where the sensor can detect the thrown objects, which is delimited by the sight of the sensor.
The robots predefined area is an area marked uo with tape on the floor. Within this area is where the robot should catch the thrown object, which is detected by the sensor. Outside the predefined area, the robot have a specific starting position, from where it will start at for each throw.

3.2 Throwing

When throwing the ball towards the trash bin, the group have two main methods in order to ensure that the sensor gets the most reliable data.

The first one being, that when throwing the ball, one must be standing with the camera/sensor to his side. From testing the Microsoft Kinect this has provided the best test results, when measuring for the distance of throw and comparing to the distance the sensor has given us.

The second method, is when throwing the ball, the trajectory curve of the ball, should have a high height so that it will cover more distance. In the future, this should not be a requirement either, but as of now, it should in order to give us more time to track, calculate and catch the object.

3.3 Detecting and tracking

The way the trash bin detects and track an object, happens when the object is of a defined form, such as a table tennis ball, which we have chosen for this project. By predefining the object the sensors should look for, the error margin becomes smaller.

In order to ensure that the sensor detect where the object is, the object has to come from a specific side of the Kinect, that provides the sensor with the best actual distance. Once again, this is just for the project as of now, and in the future, this should not affect the way detecting works, as trash can be thrown from multiple angels, depending on the location of the person and the trash bin.

3.4 Catching

Catching concerns estimating where the thrown object will fall, and then moving the robot to that location before the objects lands to successfully catch the thrown object. In order to achieve the aforementioned objective, the following steps must be completed:

Tracking the projectile trajectory to estimate a location to catch or intercept the thrown object. The tracking is a continuous process and the estimated location is transmitted to the robot repeatedly in intervals in the order of milliseconds. As the estimated location is transmitted to the robot, it will move accordingly to its relative position to the estimated location, this assumes some form coordinating system to enable the robot to move accurately. This could include first estimating whether or not the robot will even be able to reach the location before the object lands, or just move towards it regardless. It could also consider its own dimensions, as the bucket it will use to catch the robot puts restraints on how low a position the object can be, before the robot must in place to catch it, otherwise it will collide with the side of the bucket.

(As a note, could include a formula of time it takes to turn, and its forward speed.)

Moving to the location includes time to turn the robot, and the time for its wheels to power it towards the location. As the robot will continuously receive input as to the location of the thrown object will present considerations as to which degree of precision is required and to prevent jittering the robot as it makes course corrections within very small intervals. For instance if the wheels take a proportionally large amount of time to turn, and the input received informs the robot to turn every 30 ms just 1 degree which is equal to pi divided by 180 in radians, would substantially slow the robot, depending on the size of the bucket and the precision of the estimation some imprecision can perhaps be afforded.

Hardware 4

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.

4.1 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.

4.1.1 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.

4.1.2 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.

4.1.3 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.

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 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.

4.3 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.

4.3.1 Arduino Wifi 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.

4.3.2 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]

4.4 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 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 5

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

5.1 Requirements

The requirements which will be considered in this increment is mark with a blue colour.

- 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 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 with the use of trajectory prediction.

5.2 System design

The following sections describes how the marked requirements in 5.1 will be attempted to be fulfilled. The theories and ideas behind will be explained.

5.2.1 Throwing

Test have been made to ensure that the robot is provided the most time for the 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 serve the robot enough time 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 4.2 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, for the 3 meter throw, and therefore nowhere near sufficient enough.

The group decided after several tests to let the trash (still a table tennis ball) be bounced 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 in nearly the amount the double amount of time for the throw to be catched, providing an average of 2s from a 3 meter throw and 2.15s from a 5m throw.

5.2.2 Predefined area

The robots predefined area, is a field where the robot should catch the ball within. As mentioned in section 5.2.1, 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 be its starting point every time. It was set to turn a specific number of degrees and moving 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 5.1. The predefined area have an area of of 2851.5cm^2 which is 0.285m^2.

5.2.3 Microsoft Kinect

5.2.4 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

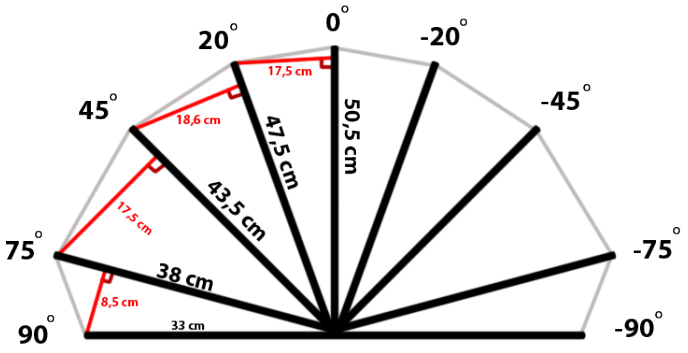

Figure 5.1. The robots predefined area

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 $9.81m/s^2$, 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* $(9.81m/s^2)$
θ : *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.

5.3 Implementation

5.4 Evaluation

6 Implementation

Tests 7

8 Discussion

Conclusion 9

Future work 10

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.

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.

Appendix A