AALBORG UNIVERSITET

Student report, not an official Aalborg University document

Embedded Systems:

Designing an Exercise Game using Dead Reckoning

Semester Project by sw507e13

from 2-9-2013 to 20-12-2013

AALBORG UNIVERSITET

Third study year
Computer Science and Software
Selma Lagerlöfsvej 300

Title: Designing an Exercise Game Using Dead Reckoning

Theme: Embedded Systems

Project period: Fall semester 2013

Project group: sw507e13

Participants:

Daníel Steínar Friðjónsson
Hamun Farzinia
Jens Møller Kursch
Mathias Winde Pedersen
Mikkel Giedsing Nielsen
Søren Skibsted Als
Toke Wivelsted

Supervisor:

Bo Thiesson

Copies: 9

Total Pages: 66

Appendix: 7

Completed: 20-12-2013

Abstract:

This project is a semester project for embedded systems, where machine intelligence has been used to track movement by utilising dead reckoning techniques. The goal of the project is to develop a game application, for a smartphone, that can motivate people to indirectly do exercise, using only the integrated sensors of the smartphone. While it proves harder to rely only on the integrated sensors, it gives the advantage that it can be used anywhere, as the approach does not require external information from GPS, Wi-Fi, etc.
By trial and error, it was discovered that the data from the integrated sensors were inaccurate. This inaccuracy lead to imprecise estimation of the current position of the phone, which meant the game was unplayable. In order to solve this issue, various filters and probability assessments, at different stages of the application were implemented. After the corrections were implemented, the application was more accurate, as the positioning no longer drifted. In conclusion, it has been demonstrated that it is possible to rely only on the data from the integrated sensors of a smartphone, to create a game application in which a user can perform exercise.

Foreword

This report was made at Aalborg University in the fifth semester of the Software study by the group sw507e13. The report was made as a part of the P5 project in the period 2-9-2013 to 20-12-2013. The project was supervised by Bo Thiesson, who has been very helpful throughout this project.

Daníel Steínar Friðjónsson	____________________
Hamun Farzinia	____________________
Jens Møller Kursch	____________________
Mathias Winde Pedersen	____________________
Mikkel Giedsing Nielsen	____________________
Søren Skibsted Als	____________________
Toke Wivelsted	____________________

Contents

1 Introduction

An increase in obesity and overweight has been reported in most parts of the world, which results in various problems, like health issues and government expenses [1, 2]. There are several reasons for this increase in obesity, e.g. an increase in fast-food consumption, lack of physical activity, and busy work schedules [3, 4]. This problem requires a solution, since a lot of people are struggling with obesity and the consequences it causes [5].

A tool to address this problem can be games, since they are fun and can, indirectly, make people exercise [6]. A platform for such games could be smartphones, as they are readily available [7]. Smartphones have integrated sensors which can be used to track the physical movement of the user. The smartphone can detect small movements such as rotation, and movements over large areas can be tracked by GPS. An example of a game can be *Cops and Robbers*, where the player is chased by other people or a virtual enemy. Another game can be *Pong*, which requires less physical space to play, but where the user moves from side to side to control a paddle.

Since people have their smartphone with them, they can play a game to exercise during breaks at work, or after spending most of the day sitting in front of a desk. This game could track and motivate a person to jog or ride a bike from and to work. If people, by playing the game, would exercise more on a regular basis, it could help reduce the overweight problem.

2 Problem Analysis

In this chapter the problem will be analysed and clarified in order to develop an application, which successfully implements a set of requirements. How to motivate people to exercise, using a smartphone application, will be analysed. To help analyse this problem, existing systems, technologies, scenarios, and the target audience, will be evaluated and specified. The results, from the analysis, lead to a final problem statement with limitations and requirements to help design the application.

2.1 Initialising Problem

How can people get motivated to start and continue exercise?

Many people are in need of exercise, but are often missing the motivation to start or continue exercising. Exercise can be fun and appealing, using motivational factors, which keeps people interested. Such motivational factors can be found in existing systems, such as games.

2.2 Existing Systems

To get inspiration for this project, three existing systems have been examined. They are called *Endomondo*, *Zombies, Run!*, and *Moves.* The following examination of the applications acts as a source of inspiration throughout development of the project.

2.2.1 Endomondo

Endomondo is a fitness tracker combined with a social network solution, which runs on smartphones. When using Endomondo, an activity is selected, such as running or bicycling. When an activity is started, the route is tracked. The tracked information available is as follows [8]:

- Overview of the route
- Distance
- Duration
- Average speed
- Maximum speed
- Maximum/Minimum altitude
- Total ascent/descent
- Calories burned

All of this information can be shared on the Endomondo website and other social media.

2.2.2 Zombies, Run!

Zombies, Run! turns normal runs into small adventures in a post apocalyptic zombie world [9]. The adventure is narrated by different characters met in the storyline. In the game the player is called *runner 5* and is part of a group of runners, who run outside the safety zone of *Abel Township* to collect resources for the town to ensure its survival. When outside the city the runner can be spotted by zombies, this event is called *Zombie Chase* which is an optional feature. If the runner is spotted by a zombie he has to run 20% faster over a period of one minute to escape. When the user is not exercising, he can play a game where the resources are spent to improve Abel Township. The user can choose to have the route tracked by GPS or the steps determined by use of the accelerometer.

Six to Start, which is the company that created Zombies, Run!, provides a website with further information about the application [9]. When a user creates an account for the website, he can synchronise the data from his phone with the website. The website presents additional data collected from runs which are not visible in the application, a sample of the information present on the website is as follows:

- Overview of the route
- Distance
- Duration
- Episodes completed
- Average speed
- Maximum/Minimum speed

Having the information available on the website is an optional feature that allows the user to track his progress.

2.2.3 Moves

The application Moves [10] uses the GPS and accelerometer for registering and counting steps taken by the user. According to their website, Moves will distinguish between walking, running and bicycling, as long as the user keeps the smartphone in a bag or a pocket. The user can give Moves information about his weight, height, age, and gender. With these physical characteristics, Moves can calculate the amount of calories burned throughout the day. Moves has a set of features that allows the user to view a summary of his tracked progress, as well as the route that he travels throughout the day. However, it is not possible to export data from the Moves application itself to other devices, but it is possible to utilise a third party application for this purpose.

2.2.4 Summary

The tracked information provided by the different applications provide some ideas of what could be recorded and shown. As an example it would be interesting to give an approximation of the calories burned. The velocity information would be more interesting in an application where it is required to run, than in an application that focuses on the user strafing back and forth or performing push-ups.

Some features from Endomondo that might be interesting for this project is the ability to share results with friends, or to compare a result with previous results in order to get motivation and to track the users progress.

In Zombies, Run! there is a story which engages the user into running. It is entertaining and motivates the user to use the application more often. Another good feature in Zombies, Run! is to view the speed of the user at any given time.

All of these features and ideas will be considered to be a part of the application for this project. Additional approaches on how to motivate a user will be explained in Section 3.1.

2.3 Embedded Systems

Embedded systems have been used for over 40 years, one of the first microprocessor was developed by Intel in 1971, and was used for business calculators [11]. Embedded systems are found in almost every household electronic, which are used in people's everyday life. They are, however, not restricted to household electronic, as they are used in industries. An embedded system consists of hardware and software, which is designed to perform specific tasks. Usually, this hardware is very restricted, in terms of memory, processing, and battery life. An example could be a coffee machine with a built-in timer function. This timer will be an embedded system, which is a very restricted system as it does not require complex calculations and memory use. While this is a very simple example, embedded systems can also be complex, such as in a smartphone or a car with multiple embedded systems that communicate amongst each other. Many embedded systems are real-time systems as they require immediate response, an example of this could be ABS in a car where response is needed immediately, as it would otherwise be a safety hazard rather than a safety function.

2.4 Movement Tracking

In this section, commonly used positioning systems available for modern smartphones will be examined. The knowledge from the examination will be used to get a better understanding of how indoor and outdoor positioning works.

2.4.1 Global Positioning System

Global Positioning System (GPS) is a system developed by and for the US military, but has proven to be highly usable in civilian applications as well. GPS is mainly used for navigation and tracking, and is able to pinpoint a receiver's location to an accuracy of a few meters. For the system to function accurately, the receiver must have signal from at least four orbiting satellites. However, when a user enters a building or a tunnel, he is unable to maintains a signal, and thereby the system is unable to function properly. In recent years, research and studies have enabled GPS-receivers to function with only three satellites. It is accomplished by looking at previously collected data, using other available sensors and basically make an educated guess or estimate the receivers current location, which is called *dead reckoning* [12]. If dead reckoning is used in conjunction with the GPS, the accuracy is increased. [13–16]

2.4.2 Dead Reckoning

Dead reckoning (DR) is a technique used to calculate the position of a device, which for this project is a smartphone. DR uses the previous stored positions and calculates the next position based upon estimated speed, elapsed time, and the direction itself. DR is commonly used for the purpose of indoor positioning and tracking. DR combined with GPS is known to be more accurate than using GPS alone, this is done by utilising inertial measurement unit (IMU) [17, 18]. Some sensors that can be used for DR are the accelerometer, gyroscope, and compass as it is integrated in various smartphones. DR is also used for ships and aircraft [19], but the focus will lie on smartphones and therefore the description of the DR techniques hereafter is applicable to smartphones.

DR can be implemented as a step length estimator, where the accelerometer serves as a pedometer, and the direction of the smartphone can be determined by use of a gyroscope and magnetometer. However, a few problems exists when using DR. The first problem is that sensor data can be flawed. For example the object could jump in place resulting in the accelerometer, gyroscope, or the compass data being misinterpreted. It can lead to false positions that are analysed for the next step.

A central problem with DR is the measurement uncertainty of the position, since various factors can disturb the calculated position. Some of the issues could be how the device is held, step size, and direction of movement. These uncertainties can prove to be inaccurate, since if one step is calculated wrong the next step will include the error. To minimise the measurement uncertainties, different constraints can be applied. These constraints can be holding the device in a specific way, restricting movement space, and limiting the movement pattern.

To summarise, GPS gives an accurate position when receiving signal from four or more satellites. DR can be used in conjunction with GPS and maintain the position when the signal to satellites are temporarily lost. However, DR can also be used without GPS, but is prone to errors.

2.4.3 Wi-Fi Position

Wi-Fi positioning techniques works by having access to one or more Wi-Fi access points. Depending on indoor or outdoor positioning there are different techniques in finding the position of the user [20].

When using *fingerprinting*, the received signal strength (RSS) from access points is recorded and a radio map is constructed from this data.

It means that a map of the environment is created with fixed points, containing RSS values of the access points, strategically scattered throughout the map. Each point has a value of RSS from a number of Wi-Fi access points, these RSS values are then used to calculate the location of a device using fingerprinting and the radio map. There is no need for hardware modification to make fingerprinting work and it is accurate enough to place people in the correct rooms in a building. However, creating the radio map can be a very time consuming process, due to the complex preparation of creating the radio map. The technique works both outside and inside, because the technique only needs the IDs of the relative RSS values of the access points in the radio map.

Angle of Arrival (AoA) is a method that uses the signals from access points and determines a location using geometry, based on the location of the access points. The device needs a special antenna, which needs to be able to calculate the angle in which the signal is received. With two or more Wi-Fi access points, the position can be calculated by using the angles from which the Wi-Fi signal is transmitted. *Triangulation*, the use of trigonometry and geometry, can be used to further enhance the accuracy of AoA. The disadvantage of AoA is the need of special hardware antennas. The advantage is that it only needs two signals from access points to calculate a position.

Cell Identity (CI) is a technique where the positions of access points needs to be known. It is assumed that the strongest signal is from the closest access point. Using this information, it is assumed that the device is close to the strongest signal and an approximation of the position is determined. It can give bad results when used indoor because of obstacles blocking the signal, and thereby making an access point further away have the strongest signal. An advantage is that CI gives a fast initial approximation of the position.

2.5 Target Audience

As previously mentioned in the introduction, Chapter 1, this application is aimed at people who need to be motivated towards physical activity. It is a broad audience and includes people who are obese or in need of regular physical activity. A requirement is that they need a smartphone, which is a common device today. The age range of the target audience ranges from young kids to middle aged people who like games.

Income can be ignored, as the the target audience is already limited to people who have a smartphone. The application will not be aimed at a specific gender as obesity and the need of physical activity impacts both genders. Furthermore, smartphones are equally used by both genders [21].

2.6 Scenarios

In this section, examples of how this application can be used will be examined. The areas that need to be examined are indoor and outdoor exercises. There will be created two scenarios with the characters John and Brian.

2.6.1 Indoor Exercise

John has an office job where he spends most of his days sitting in a chair. His work schedule is often full and therefore he rarely gets any exercise. John acquires a new application which should help him with his exercise during his work schedule. Every now and then he takes a short break from his work and by the help of the application he gets some exercise. After using the application frequently, John feels much more energetic.

2.6.2 Outdoor Exercise

Brian requires some exercise because he is out of shape and obese. Obesity is a problem for Brian as it is becoming a serious health issue. Brian uses an application to start his exercise. The application tracks his exercise and notifies him of his progress, which in turn motivates him to continue doing exercises. Brian shares his progress with his friends, who then join him in his next exercise session. After a couple of months of exercising, Brian has lost weight and his health situation has substantially improved.

2.7 Problem Statement

After analysing the topics in the sections above, a problem statement can be declared. Systems already exists, which help people with their exercise, either by tracking various information about an exercise, or motivating them e.g. with a story. Most of these existing applications utilise GPS to track the users' exercise, while a few of them apply a dead reckoning technique that allows both indoor and outdoor exercises. For that reason it is believed it would be beneficial to make a smartphone application that is not restricted to being outside. This decision is chosen, since the amount of applications that offer indoor activities are limited, compared to the selection of GPS based applications.

With these previous sections in mind, a hypothesis can be stated:

A game that is fun and motivates the user to exercise can be developed for a smartphone using dead reckoning.

To help verify this hypothesis, the following questions need to be answered:

1. What are the requirements for a smartphone application that make people want to exercise?

2. How can a smartphone application motivate people to exercise?
3. Why would people use this application instead of other applications?

2.8 Limitations

As the development time is limited, some limitations have to be declared before the requirements are specified. The application is limited by the sensors of the smartphone, what movements they are able to read, and how accurate the application is. Another limitation is the implementation language for the application, which in this case is C#. The language restricts the platforms that can be used, but C# is still a versatile language.

As mentioned in Section 2.4.2, dead reckoning is hard to implement, without some limitations of various forms, due to measurement uncertainties. For this reason, limitations to how the user is allowed to operate the device has to be specified. To play the game, the user is limited to holding the smartphone in a fixed position in front of him, and has to look at the display. Another limitation is how the user should move, which will be limited to left and right movement.

Another important limitation for this project is the allotted time, which is limited to a single semester. Having limited time, will not allow for the implementation of the application for multiple platforms such as iOS, Android, etc. For this project the application will be developed for the *Windows Phone 8* platform, using a *Nokia Lumia 820.* Nokia Lumia 820 provides the essential sensors for dead reckoning and does not limit the movements that can be registered.

2.9 Requirements

In this project there is a set of requirements that should be met. The purpose of the project is to develop a game which makes users exercise, tracking the movement with dead reckoning. The hard and soft requirements for the application will be described and reasoned for hereafter.

2.9.1 Hard Requirements

The hard requirements are the essential requirements of the application.

Exercise

The application to be developed is meant to be a game application with exercise in mind. Therefore, physical activity is required from the user. Since the user has to walk left and right in order to play the game, this is considered cardiovascular exercise. While there are many different forms of exercise, the focus of this application is to encourage physical activity to lose weight. However, various forms of cardiovascular exercise would be a great addition to the application, but due to the time constraints of this project such exercise variations are not pursued.

Dead Reckoning

As previously mentioned in Section 2.4.2, dead reckoning is able to track a user's movement more precisely on a small scale, compared to the GPS and works indoor. Understanding and implementing dead reckoning is a key goal of the project, as the intent of this semester is to acquire knowledge of machine intelligence. Dead reckoning will be used in the game to control the movements inside the game. These movements will, as mentioned, make the user physically active. Another result of using dead reckoning, is that the game will be playable everywhere without relying on non-embedded hardware.

Game

The final application should be a game that includes all the hard requirements, seen above. The game has to be controlled by the player, whom can move in one axis, meaning that a player can move both left and right. The game type should be similar to Space Invaders, Pong, and Break-It. One of the key points of making the application a game, is to motivate people. The requirement includes the game, itself, while additional motivational features, such as high scores, are not included in this requirement.

2.9.2 Soft Requirements

If the application has met the hard requirements and there is enough time, the soft requirements can be implemented.

Two Dimensional Movement

Two dimensional (2D) movement is an extension to the movement allowed and needs adjustments in the dead reckoning techniques. If it proves to be easy to develop control for one axis, the scope of the project will be extended to 2D movement. Which means if 2D movement is implemented, controls will allow for other types of games, e.g. moving in a labyrinth as in Pacman, which will result in other movement patterns.

Motivational Features

Motivational features can be implemented to make the user start and continue playing the game. Some motivational features could be high scores, in which users can compare and compete amongst each other, resulting in increased replay value. Other features that can affect the replay value could be achievements in the game as well as different levels or difficulties. Additional motivational features can be seen in Section 3.1.

3 System Analysis

In this chapter the required components for the system are analysed. First a discussion of different motivation factors will be described, which gives a foundation for the system components. The components include the tools which are used to create the game and the sensors which are essential for utilising dead reckoning.

Lastly, a system definition will be written using a FACTOR analysis, which is an Object-Oriented Analysis and Design tool [22].

3.1 Motivation

Motivation is good when starting to exercise, because after the first couple of weeks people tend to lose motivation and stop exercising. There are many factors to keep a user motivated. These factors affect people differently and it is therefore near impossible to get everybody motivated. However, instead of motivating people to do exercise, a fun game can be made, that people like to play and indirectly, by playing the game, physical activity is achieved. Good motivation leads to frequent exercises, since it is fun doing them [6].

Motivation can be amplified by making the game able to track the user's exercise and compare it with others. It does not necessarily have to be how hard the user has exercised, but instead how well they performed in the game, as well as other ways to make the game entertaining and interesting.

To elaborate more on the approach for motivation that will be used in this project, various factors can be considered when making a game that is fun, immersive, and addictive. The game is meant to motivate people to indirectly do exercise. It will be a game with controls as in old arcade games such as *Falldown, Break-it, Space Invaders,* and *Pong*, where the user, in this case, is the controller.

Apart from the mentioned properties, other features could be a high score, so the user is able to see if he is better than his friends, which can keep the competitive users motivated. A high score can, arguably, be de-motivational if the user is compared to people with a higher score. On the other hand, it can also be motivating for people who are not giving up, since they want a better score than their friends. Generally, people like to be rewarded for their progress, this reward could be upgrades. When making upgrades, it is important that the upgrades are achievable in a reasonable amount of time. Furthermore, since the game attempts to get the user to exercise, the upgrades should not lower the amount of exercise that the user gets. Therefore, upgrades could be additional movement, such as, jumping, moving forward etc. to add additional exercise patterns.

3.2 Game Development Tools

One of the hard requirements, seen in Section 2.9, is the development of a game. In order to ease the development of this game, two game development tools will be analysed. The target platform of this project is Windows Phone 8, therefore the tools have to work on this specific platform. There are many different tools for developing games on the Windows Phone 8 platform, however, in this report the focus is on XNA and Unity. These tools have been selected due to their popularity as well as documentation.

3.2.1 Unity

Unity is a software game engine in which it is possible to create a game, where the engine provides functionalities. Unity has a lot of available features for creating a game, such as game mechanics, 3D modelling environment, and a lot of existing scripts to ease the game development. While a lot of the mechanics is predefined it still remains possible to edit and create mechanics such as physics within a game.

There are three available programming languages in Unity, namely C#, JavaScript, and Boo. Unity uses .NET 2.0 according to Xamarin [23]. Because Unity uses *monoDeveloper 2.8.2* as its main IDE, Unity uses C# version 4.0 [24]. It means that some features that are dependable on a .NET framework higher than 2.0 will not work in Unity. When programming an application in Unity it can be ported to a wide range of platforms, which include Android, iOS, and Windows Phone.

3.2.2 XNA

XNA is a software game development framework in which it is possible to develop games. The XNA framework supports the developer with libraries that allow the developer to construct a game, where the game mechanics are written by the developer. Due to XNA being a framework, it does not have a user interface.

When programming in XNA the programmers are able to program in C#, Visual Basic, F#, JavaScript, and C++. XNA is a Microsoft product and can be ported to any Microsoft platform. XNA has some built-in features that makes it easy to make games, such as network communication. XNA runs on the .NET framework 4.0 [25], which means that XNA can use features supported by the .NET 4.0 framework.

3.2.3 Summary

Both applications provides a variety of features for creating games. When it comes to game development, both of them are equally fit, as obtaining information from the sensors requires a single instruction [26, 27]. Unity offers a user friendly 3D environment whereas XNA is purely textbased and therefore less user friendly. Since both of the applications offer C# as a programming language, this means potentially both could be used for the project, corresponding to the chosen development language specified in Section 2.8. Both applications supports various target platforms, but Unity has a wider variety of platforms the application can be ported to. However, the variety of supported platforms is not a factor for this project as the project is for Windows Phone 8. In conclusion of this it has been decided to use XNA because it uses a newer version of .NET framework, and since an extensive visual platform is not needed to create a simple game.

3.3 Sensors

The game for this project is intended to make people exercise, and it is therefore necessary that the game can track the users' movement. As a result, it is crucial that the platform, which the game is running on, has the sensors needed. Fortunately, this is the case for many smartphones. The necessary sensors to track the users' movement will be thoroughly examined.

3.3.1 Accelerometer

An accelerometer is a sensor that is used to track the acceleration of a given device and the information tracked is represented in three values. The three values are the acceleration in the x-, y-, and z-axis. The accelerometer can simulate a pedometer, where the differences in the acceleration values can be recognised as a step, this technique can be used in dead reckoning. The difference between a pedometer and an accelerometer is that the pedometer can only count steps, where the accelerometer readings can be used to calculate how far the steps are. Furthermore, the accelerometer is not limited to function as a pedometer and can use the tracked acceleration for other calculations such as game input. For this project the accelerometer can be used to track the user's movements.

3.3.2 Gyroscope

A gyroscope is a sensor that is used to track the orientation of a given device, this information is represented in three values. These values are the orientation in the three axes. The gyroscope can be used as a compass, to do this the device has to know precisely which way is north before it can calculate which way the device is orientated. However, the measurement uncertainty can become a great concern the more the device is turned. The gyroscope can be used when creating dead reckoning algorithms, since it can calculate which way the device is orientated to determine when the device is turned.

3.3.3 Magnetometer

A magnetometer is a sensor which is used to track the magnetic field of the earth, but can also be affected by other magnetic fields. The magnetometer can show how the device is orientated, compared to the magnetic poles. The information can be used to track if the user is turning around. The magnetometer can help adjust the measurement uncertainty of the gyroscope.

3.4 FACTOR

FACTOR is a tool to provide a structure for the system definition [22]. The different topics that FACTOR covers will be described hereafter.

Functionality

A game where the user's movement is used as input to play the game. An elaborate description of the game will be explained in Section 4.5.

Application Domain

The product will be a smartphone application, which can be used both indoor and outdoor with sufficient space to move around. The smartphone application will be developed for a Windows Phone 8.

Condition

Usable for people owning a Windows Phone 8, as described in Section 2.5.

Technology

The application will run on a smartphone with Windows Phone 8, chosen to prove the concept, as the developers had experience with C#. Furthermore, the smartphone is required to have an accelerometer, a gyroscope, and a magnetometer. To track the user's movements, dead reckoning will be used. The tool used to develop the game is XNA, as discussed in Section 3.2.

Objects

The elements typically present in old arcade games will be represented as objects. For example, the ball and paddle in Pong.

Responsibility

The application has to track the movement of the user in a sufficiently accurate way, so the user does not move several steps without the application recognising it, which is very important for a reliable gameplay.

4 Theory & Design

In this chapter, the theory and design for this project will be described and specified. The theory will cover position calculation techniques, Bayesian networks, and normal distribution. Afterwards, the machine intelligence theories of this project are specified, game and system design will be reviewed using event tables, state diagrams, and illustrated concepts.

4.1 Position Calculations

In order to develop a game where a user moves to control the game, it is necessary to register these movements and calculate a position. Calculating positions can be done by analysing the data given from the accelerometer in the phone. The output of the accelerometer is the accelerations of the three axis' of the accelerometer. Figure 4.1 and Figure 4.2 shows two graphs with each graph showing three steps. Figure 4.1 is the acceleration along the y-axis of the phone. Figure 4.2 is the velocity along the y-axis of the phone. The velocity is calculated by taking the integral of the acceleration, the formula is as follows:

$$v(T) = \int_0^T a(t)\,\mathrm{d}t$$

where, $v(T)$ is the velocity at time T, and $a(t)$ is the acceleration at time t.

A graph with accelerations and a graph with the corresponding velocities can be seen in Figure 4.1 and Figure 4.2, respectively.

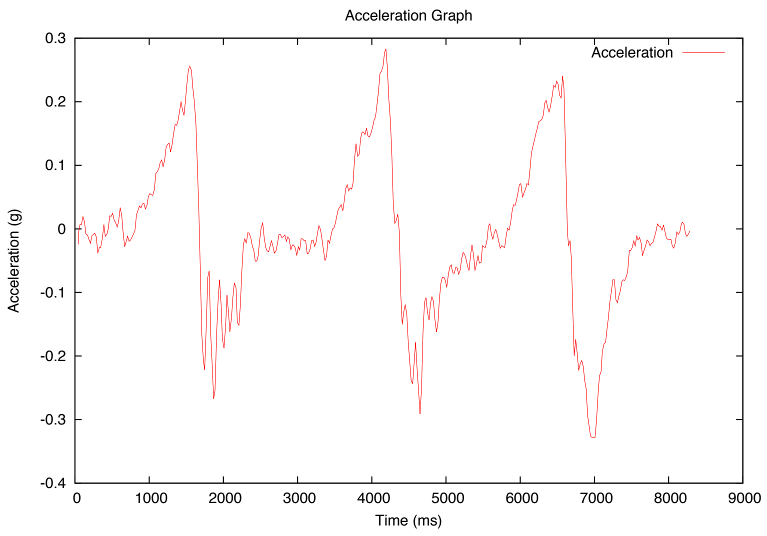


Figure 4.1: Acceleration readings over three steps.

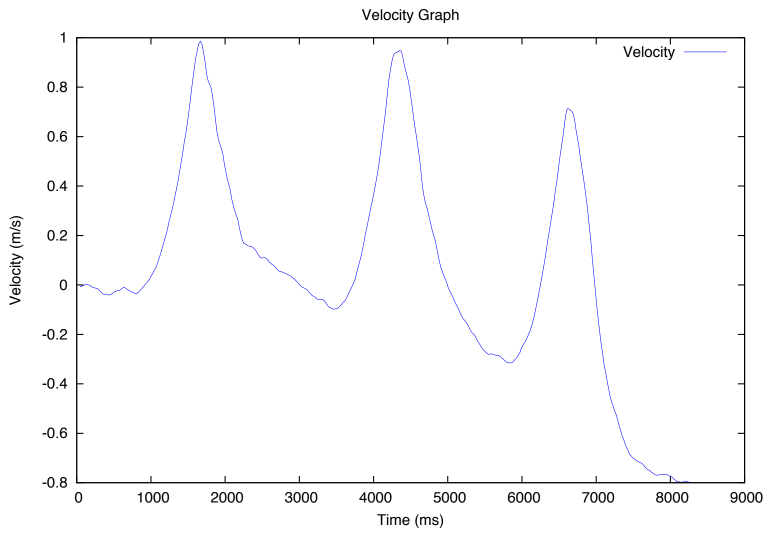


Figure 4.2: Velocity calculations over three steps.

The velocity graph is smoother than the acceleration graph because each point on the velocity graph is a summation of the area of the acceleration graph.

To determine the position, the area under the velocity graph is calculated by taking the integral of the velocity over time, as seen in the following equation:

$$s(T) = \int_0^T v(t)\mathrm{d}t \tag{4.1}$$

where, $s(T)$ is the distance at time T, and $v(t)$ is the velocity at time t.

The value can be used in conjunction with the previous position to calculate an approximation of the new position. The actual values can be observed by video footages of people sidestepping with the phone. As seen in the graphs, noise exists and should be corrected by use of filters.

4.2 Filters

Filters are needed to cancel out noise from the acceleration inputs. Noise is caused by electronic noise from the circuitry and mechanical noise from the sensor itself [28].

4.2.1 The Kalman Filter

The Kalman filter is built upon a simple found propagation in a dynamic Bayesian network, where the previous state and new observations is used to find the current state. A Bayesian network can be used to find the probability distribution of the next state given the current state. The Kalman filter is a linear dynamic system, which means the function derived will be a linear function.

The Kalman filter structure can be modelled as a dynamic Bayesian network, see Figure 4.3. Insertion of evidence in the network will be described in Section 4.4.4. Additional theory for Kalman filters exists, but has not been examined, as it was unnecessary for this project.

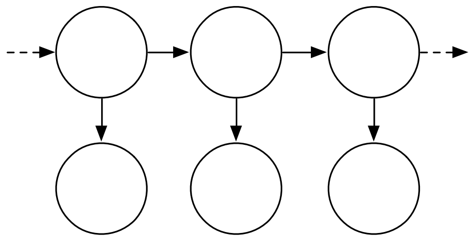


Figure 4.3: The Kalman filter.

4.2.2 Exponential Moving Average

The exponential moving average (EMA) reduces noise from the input and smoothes the data set. The formula for exponential moving average is as follows:

$$EMA_i = \alpha * y_i + (1 - \alpha) * EMA_{i-1}$$

where,

EMA_i	is the exponential moving average value for the i'th data element.
α	is a coefficient which determines the smoothness of the data.
y_i	is the i'th observed value.

EMA_i is calculated recursively, updating the values. The previously observed values are accounted for in the formula by EMA_{i-1} but their impact is reduced in each step. The formula works by weighing the new value and adding it with the weighted previously observed values EMA_{i-1}. Figure 4.4 shows two graphs where the red graph is the raw data and green is the data after running the exponential moving average, with an α of 0.1.

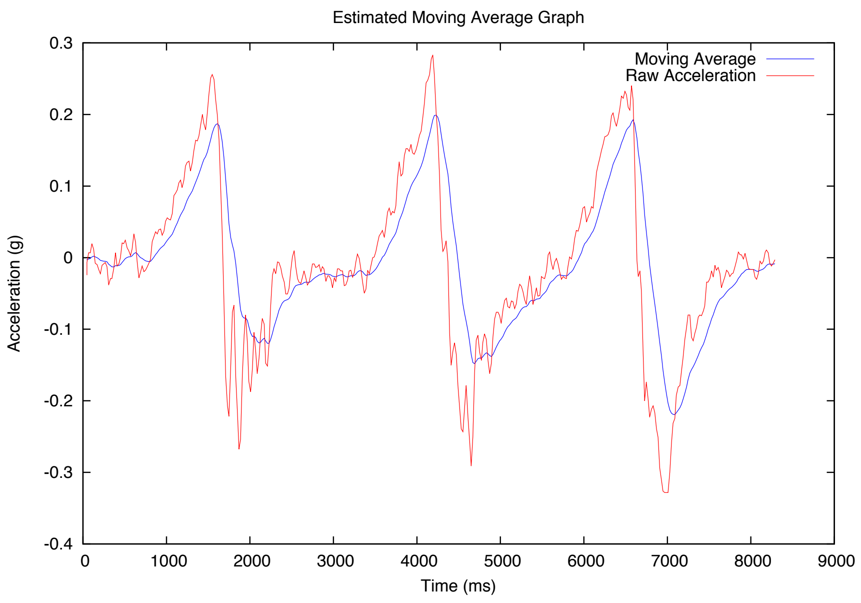


Figure 4.4: Raw data, red graph, compared with the EMA data, blue graph.

4.3 Bayesian Network

A Bayesian network is a directed acyclic graph, where each edge shows how a parent has an influence on its child. The Bayesian network, see Figure 4.5, shows how the different variables affect each other. In the figure there are three different types of variables. The first variable A, which is shaded, is an information variable. Information variables, can be either stochastic or deterministic, hold information that are obtained from input-sources. The second variable B is a deterministic variable, marked with two circles, the data in this variable type is based on parent values, and has no probability distribution. Lastly, the variable C with a single circle, is called a stochastic variable. Stochastic variables can be either continuous or discrete. Continuous variables can hold an infinite number of values between two points, while discrete is a finite set. As an example, a continuous variable holding a value between two points, 0 and 1, can have any number between these two points, which is an infinite set of real numbers.

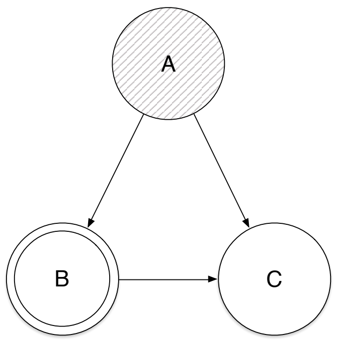


Figure 4.5: A simple Bayesian network

When constructing a Bayesian network the following must be considered:

- What are the relevant variables?
- What values types should be in the network?
- What is the relationship between the different variables?

4.3.1 Dynamic Bayesian Network Design

With the types of variables described, a dynamic Bayesian network, Figure 4.6, and the variables dependency herein will be explained.

Figure 4.6, is divided into timeslices, a new timeslice is created when the sensors registers a new value. All the variables in the dynamic Bayesian network are stochastic continuous variables. The independent variables in the network are the *Acc* and *Gyro* variables, which are the information variables. The mean values are read from the accelerometer and the gyroscope respectively.

The *Atan2* variable transforms acceleration values into angles, explained in Section 5.2. The *Comp*, short for Complementary Filter, variable contains angles obtained by utilising the complementary filter, which is dependent on two angles from the Gyro and Atan2 variable, explained in Section 5.2. *SF*, short for Sensor Fusion, serves the purpose of combining the collected data and output a corrected acceleration. The *MA*, short for Moving Average, variable takes the calculated acceleration value and applies a moving-average filter to it, as described in Section 4.2.2.

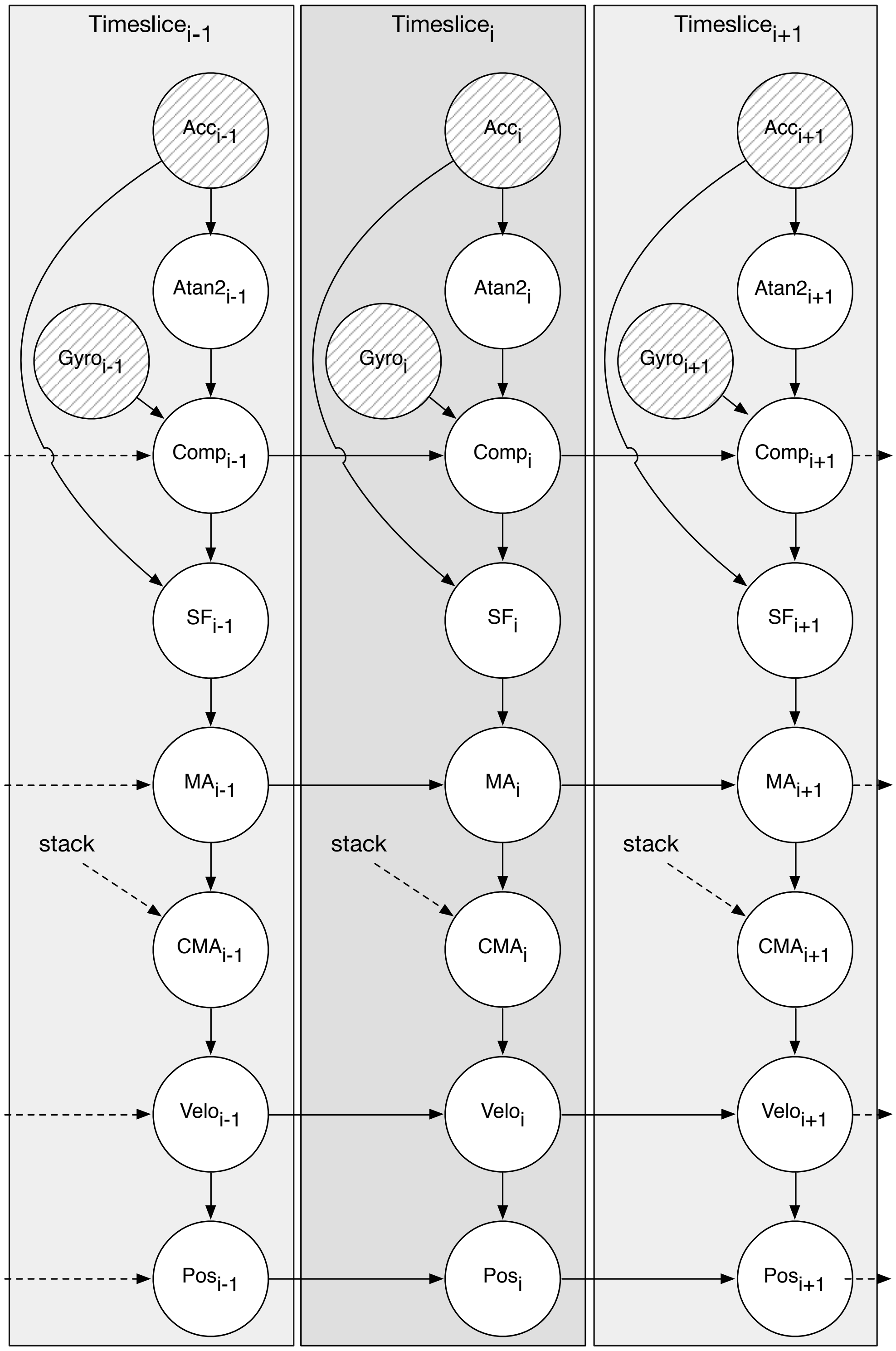


Figure 4.6: Dynamic Bayesian Network

To make sure, every acceleration is followed by an equal deceleration, the *CMA*, short for Corrected Moving Average, variable is implemented. The purpose of CMA is to save MA variables in a stack, so that once deceleration begins, the application will pop elements from the stack and use these values negated.

The *Velo* variable calculates the current velocity by using the values obtained from the CMA and previous Velo variables. Lastly, the *Pos* variable calculates a new position, by looking at the previous Pos and the current Velo variable.

Probability distributions will be determined as described in Section 4.4. The structure of the dynamic Bayesian network is based on the physical laws for the relation between acceleration, velocity, and position, as described in Section 4.1.

4.4 Normal Distribution

In this section, the theory needed to use normal distributions is presented and explained, and has primarily been based on Lauritzen and Jensen [29], Thiesson et al. [30].

A normal distribution is common when working with continuous variables [31]. It is a distribution that can be specified by giving the mean and variance, the uncertainty about the expected value of the distribution, notated as $\mathcal{N}(\mu, \sigma^2)$. It could be a distribution for the expected result of i.e. acceleration, velocity, or position.

The theory of univariate and multivariate normal distributions is described hereafter, where univariate is for a variable influenced by one factor, whereas multivariate is by multiple factors.

4.4.1 Univariate

The actual function for a univariate normal distribution is as follows:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2*\sigma^2}} \tag{4.2}$$

where,

μ is the mean value of the data. It gives the position of the normal distribution, which is approximated by using $\mu = \frac{1}{n}\sum_{n}(X_n)$.

σ is the standard deviation. It is a measurement for the variance of the data, and determines how flat the normal distribution is. σ^2 is the variance that is approximated by using $\sigma^2 = \frac{1}{n}\sum_{n}(X_n - \mu)^2$.

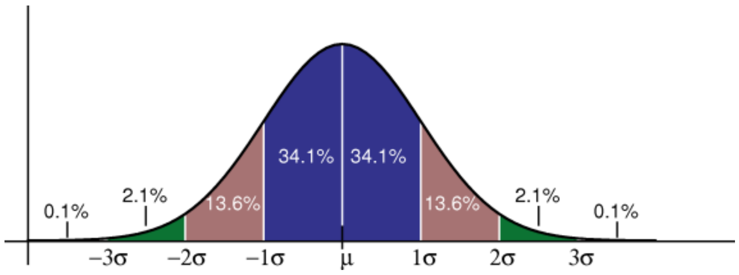


Figure 4.7: Normal distribution [32].

To have an idea of how a normal distribution looks, see Figure 4.7. The figure indicates how the probability is distributed around μ, where 68% of the values is in the range of $\mu \pm \sigma$. If the integral is taken from $-\infty$ to ∞, it will result in 100%.

4.4.2 Multivariate

To update the probability distributions for the given variables in the Bayesian network, for each timeslice, it is necessary to have some general theory for joining and marginalisations of probabilities.

The probability distribution for a stochastic vector X, where the distribution is normal and not affected by other variable distributions, is defined as follows.

$$p(X) = \mathcal{N}(A, C) \tag{4.3}$$

As can be seen in Equation (4.3), the mean value is solely based on A, which is a $\left[a \times b\right]$ matrix. Furthermore, C is a $\left[a \times a\right]$ co-variance matrix for the normal distribution of X.

When a stochastic vector Y is affected by another stochastic vector X, then the conditional density for Y given X is defined as follows.

$$p(Y|X) = \mathcal{N}(E + F * A, G) \tag{4.4}$$

In Equation (4.4), $E + F * A$ can be seen and is a linear regression on A. E is a $\left[1 \times d\right]$ gain matrix that can be defined, for example if there is a tendency to get a mean value that is always 0.1 lower than the actual value, E could be 0.1 to take this into account. A is a $\left[c \times d\right]$ matrix, containing the mean values of the parent variables of the affected variable, where Y is the child. F in this case is a $\left[1 \times c\right]$ matrix, and is defined such that it correctly describes how X affects the mean value for Y, which varies depending on how X is constructed. That is, how the different elements of A, which are based on the mean values of X, affect the mean value of Y. G is the cardinal covariance matrix, for Y given X, which for this project has not been determined accurately. Utilising machine learning techniques, could be helpful to determine the accurate G.

4.4.3 Direct Combination

The equation to calculate the joint distribution of Y and X is as follows:

$$\begin{aligned}
P(Y, X) &= P(Y|X) * P(X) = \mathcal{N}(U + V, W) \\
U &= \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix} = \begin{bmatrix} A \\ E + F * A \end{bmatrix} \\
V &= 0 \\
W &= \begin{bmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{bmatrix} = \begin{bmatrix} C & C * F^\intercal \\ F * C & G + F * C * F^\intercal \end{bmatrix}
\end{aligned} \tag{4.5}$$

As can be seen in Equation (4.5), U consist of the mean values of X and Y. When $P(X)$ and $P(Y|X)$ have been defined, the partial result can directly be used for U. No additional gain other than U is relevant since $V = 0$, as there are no conditional variables in $P(Y, X)$. The covariance matrix W for $P(Y, X)$ can, as U, also be constructed by the partial results of marginalisation X in $P(X) * P(Y|X)$. The elements of W consists of the variance of X and Y on the diagonal, where the other elements are the co-variances of X and Y combined. In order to calculate the probability distribution for Y, Equation (4.6) is used.

$$P(Y) = \mathcal{N}(E + F * A, G + F * C * F^\intercal) \tag{4.6}$$

Equation (4.6) has been constructed by using $P(Y, X)$, where X has been marginalised out. Since the probability distribution, which is being worked with, is a normal distribution, this marginalisation becomes trivial. The mean value in the equation is $E + F * A$, since it is the Y part of U in Equation (4.5). The variance is element $(2, 2)$ in W from Equation (4.5).

4.4.4 Insertion of Evidence

When able to calculate the joint distribution of two stochastic variables, it is possible to update the probability distributions with insertion of evidence, which is defined as follows.

$$\begin{aligned} P(Y,E) &= \mathcal{N}\left(\begin{bmatrix} A_Y \\ A_E \end{bmatrix}, \begin{bmatrix} C_{YY} & C_{YE} \\ C_{EY} & C_{EE} \end{bmatrix}\right) \\ P(Y,E=e) &= \mathcal{N}(U,W) \\ U &= A_Y + C_{YE} * (e - A_E) * C_{EE}^{-1} \\ W &= C_{YY} - C_{YE} * C_{EY} * C_{EE}^{-1} \end{aligned} \tag{4.7}$$

As can be seen in Equation (4.7), when evidence is inserted for a stochastic variable, the joint distribution for two variables where one of them has evidence inserted, can be determined from the evidence and the joint distribution of these two variables.

The insertion of evidence theory described above works when having an exact observation. However, if an observation is uncertain, the above formulas do not work since the variance from the observation wont have effect on the marginalised distribution of the parent node of the observation. The theory for evidence can, however, be used in other settings such as the Kalman filter, since in that case the observation is certain.

4.4.5 Applying the Formulas

With the general theory for direct combination described, applying the theory to the Bayesian network for this project can be performed.

See Equation (4.8) for an example of how to determine the normal distribution of the acceleration.

$$P(acc_{i+1}) = \mathcal{N}(acc_{i+1}, \sigma_{acc}^2) \tag{4.8}$$

Equation (4.8) consists of acc_{i+1}, which is the recorded data from the accelerometer, and σ_{acc}^2 is the variance, which is determined in Section 6.5.1. The reason the acceleration is represented as a single stochastic node and not as two nodes, where one is the observation of the acceleration and the other is the actual acceleration, is as follows. As discussed in the previous section about evidence, the variance for the observation would not have an impact on the actual acceleration node. For that reason, acceleration has been modelled as a single stochastic node to take the acceleration variance into account for its descendants.

Once this normal distribution is determined, the normal distribution for the moving average of the acceleration MA_{i+1}, as seen in Figure 4.8, can be found.

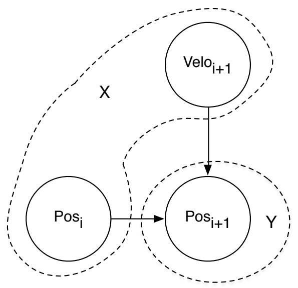


Figure 4.8: A family, or segment, of the Bayesian network from Section 4.3.1.

With the assumption of a Bayesian network as in Figure 4.8, Equation (4.9) is valid, given the previous theory.

$$\begin{aligned} P(Pos_{i+1}) &= \mathcal{N}(F_{Pos} * A_{Pos_{i+1}}, G_{Pos_{i+1}} + F_{Pos} * C_{Pos_{i+1}} * {F_{Pos}}^{\intercal}) \\ &\text{where} \\ F_{Pos} &= \begin{bmatrix} \Delta t & 1 \end{bmatrix} \\ A_{Pos_{i+1}} &= \begin{bmatrix} \mu(Velo_{i+1}) \\ \mu(Pos_i) \end{bmatrix} \\ C_{Pos_{i+1}} &= \begin{bmatrix} var(Velo_{i+1}) & 0 \\ 0 & var(Pos_i) \end{bmatrix} \end{aligned} \tag{4.9}$$

In Equation (4.9), F_{Pos} is based on the formula for exponential moving average, as seen in Subsection 4.2.2. A_{Pos} is based on the mean values for the parent variables of Pos_{i+1}. C is a $\begin{bmatrix} 2 \times 2 \end{bmatrix}$ matrix and is constructed from the variance of the parents. $G_{Pos_{i+1}}$ is a cardinal covariance matrix for Pos_{i+1} given a stochastic vector consisting of $Velo_{i+1}$ and Pos_i.

For a complete set of formulas for the Bayesian network in Section 4.3.1, derived from the theory in this section, see Appendix A. The formulas are excluded from this section as they are derived similar to Equation (4.9).

In principle, to propagate through a family, the process is to first calculate the marginal and conditional distributions and then make a joint distributions for these, whereafter a marginal distribution for the child is found.

4.5 Game Design

The intended design of the game will be similar to *Squash*, while retaining the graphics of *Pong*. In Figure 4.9 the layout of the game board can be seen. The basic elements of the game are the ball, the paddle, the walls, and power-ups. The main objective of the game is to prevent the ball from moving past the paddle.

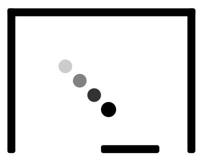


Figure 4.9: The layout of the game board

An important game design aspect is the control of the paddle, which in this project relies on dead reckoning for movement tracking. It means if the player moves left or right, the paddle will follow his movements.

While the basic elements of the game are simple, some additional features have been added to the game to enhance the user experience. Some of these features include a scoring system and power-ups. The score is based on the time the player keeps the game alive, meaning as long as there is at least one ball on the board, the score increases. Power-ups could increase the size of the paddle, get an extra ball on the board, or slow the speed of the balls, and negative power-ups could decrease the paddle size, increase the speed of the balls, or decrease the score. A story board of how the game should work can be seen in Figure 4.10.

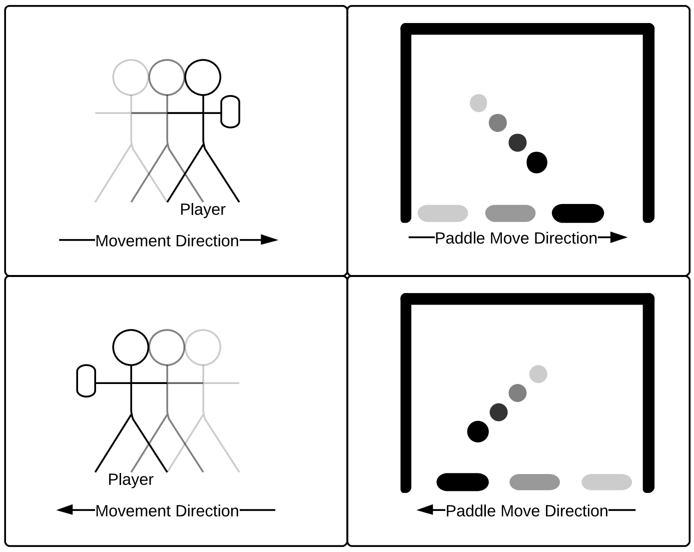


Figure 4.10: A storyboard of how the game is played

4.6 Events

The event table in Table 4.1 represents the possible events in the game and how they are connected to the classes. The classes used are *Paddle*, *Ball*, *Time*, and *Score*. The Paddle class ensures that the movements of the player corresponds to the paddle in the game. The Ball class represents the ball which moves on the board, and can bounce off the walls and the paddle. Time starts when the game starts and stops when the ball is not caught. Time is also used to limit the duration of power-ups and increases the score as time passes.

	Paddle	Ball	Time	Score
Lose game		+	+	+
Move	*			
Spawn ball		+		
Despawn ball		+		
Spawn power-up			*	
Change paddle size	*			
Increase ball speed		*	*	
Decrease ball speed		*		
Increase score			*	*
Score multiplier				*

Table 4.1: Event Table, * meaning zero to many and + meaning zero to one.

Lose game is an event that happens when the ball is not caught by the paddle. When the event occurs, the time is stopped and the Score is saved in case the user wants to submit it to the high score. To avoid losing the game, the paddle has to be able to move. The *Move* event works together with dead reckoning to get the movement of the player, and enables the paddle to move left and right.

Spawn power-up is an event that spawns a power-up in the game, which occurs randomly after some time. One of the power-ups that can spawn is a power-up that increases the paddle size and the counterparting power-up decreases the paddle size.

The ball spawns when the game starts, and despawns when the ball is not caught. This is handled by the *Spawn ball* and *Despawn ball* events. The ball speed is increased over time, but can also be changed with the help of power-ups, which is the *Increase ball speed* and *Decrease ball speed* event.

The score can be affected by two different events. *Increase score* happens over time, whereas *Score multiplier* occurs when its corresponding power-up is caught, resulting in either a doubling or halving of the score, depending on the power-ups colour.

4.7 State Diagram

A state diagram shows the different states the program can be in and the options the user can perform to change the state of the program. State diagrams are useful models for designing the interfaces. The state diagram for this project can be seen in Figure 4.11, which includes all the transitions and states for the program.

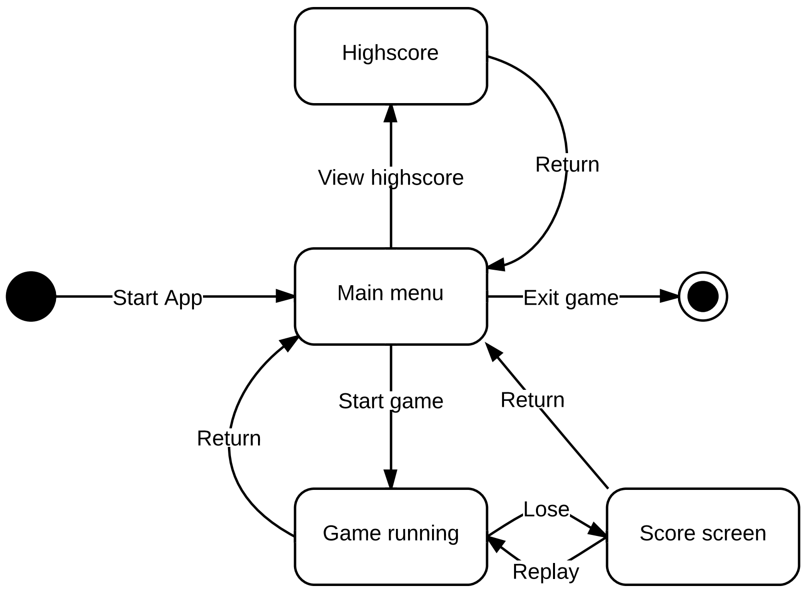


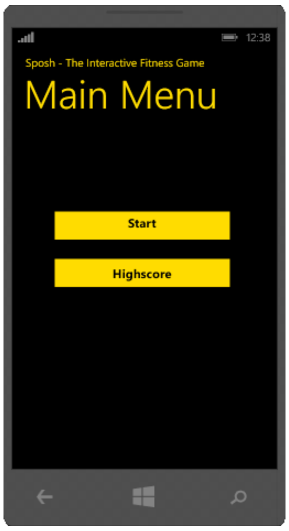
Figure 4.11: Application state diagram.

Initially, the user will launch the application, which will present the *Main menu*. From here, the user can press the buttons *Start game*, *View highscore*, or *Exit game*. Start game will render the game to begin, and the application will now be in the state *Game running*, it will remain in this state until the user loses the game, or the user chooses to *Return* to the Main menu. When the user loses the game, a *Score screen* will appear, allowing the user to *Replay* or Return to the Main menu. Furthermore, when a user loses a game, the achieved score can be submitted to the highscore, which can be viewed in the *Highscore* state. The Highscore state can be accessed from the Main menu.

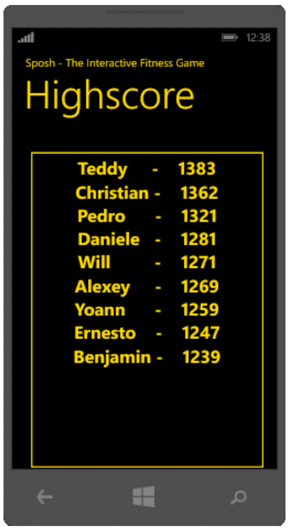
4.8 Concepts

In this section concepts will be illustrated. The concepts will be used as a draft before implementing the final application, where the state diagram Figure 4.11 is used to derive the layout.

The user is first presented with the *Main Menu*, which can be seen in Figure 4.12a. The Main Menu will enable the user to start the game and view the *High score*. In the High score, Figure 4.12b, the user can see a list of scores achieved on the phone.



(a) Main Menu

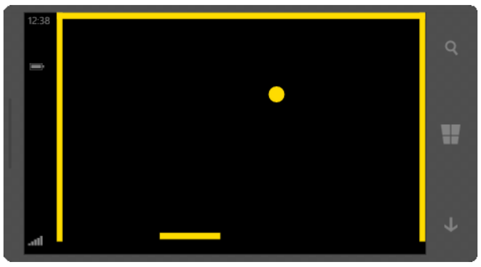


Teddy - 1383
Christian - 1362
Pedro - 1321
Daniele - 1281
Will - 1271
Alexey - 1269
Yoann - 1259
Ernesto - 1247
Benjamin - 1239

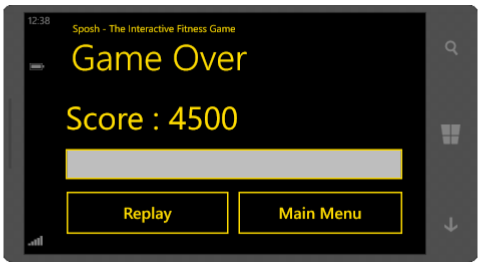
(b) High score

Figure 4.12: The prototype of the Main Menu(*a*) and the High score (*b*)

In Figure 4.13a, the user plays the game. In this screen the borders of the game, the ball, and the paddle can be seen. When the ball has passed the paddle, the user loses the game and is taken to the score screen, where the user can save his score, replay the game, or return to the main menu, as can be seen in Figure 4.13b.



(a) The Game.



(b) The Score Screen.

Figure 4.13: The game (*a*) and the score screen (*b*)

5 Sensors & Data Collection

This chapter describes the measurement and collection of readings to determine the orientation of the accelerometer, gyroscope, and magnetometer. After determining the orientation of the accelerometer, additional data of the user movement is recorded. Hereafter, the gyroscope and the magnetometer are described, and readings from these sensors are used to determine the angle of the device. The data will be examined to determine how the movement of the user affects the acceleration.

5.1 Accelerometer

Before starting the implementation of the application, the accelerometer of the phone should be examined. Knowing the orientation of the accelerometer axis in the phone is important in the implementation of dead reckoning. As mentioned in Section 2.8, the phone will be in held in a fixed position when the game is played. Which means that sideways movements will be tracked along one axis, and will be used to move the paddle accordingly. To find the axis orientation, tests have been made by moving the phone in specific directions and looking at the data output. The orientation of the accelerometer axis' can be seen in Figure 5.1.

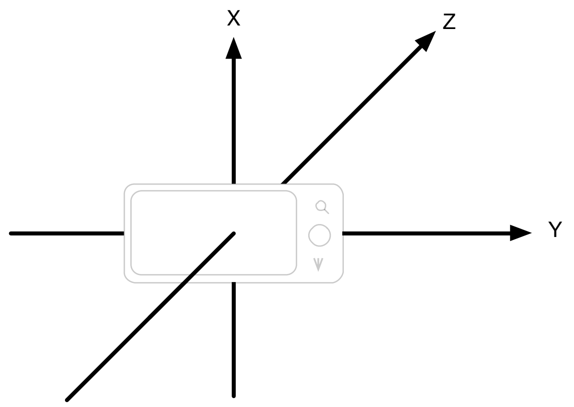


Figure 5.1: The orientation of the accelerometer.

Another functionality that needs testing is how the data behaves when the phone is rotated manually around the different axis', which is needed because when the user holds the phone it will be tilted. In Figure 5.2, a graph shows the accelerations of the axis when the phone is rotated counter-clockwise around the y-axis from the position shown in Figure 5.1, other recordings was performed for rotation around the other axis' but has been omitted as it does not give new insight to the understanding of the accelerometer. When the user is holding the phone as in Figure 5.1, the accelerations, in the device's y-axis, of a step to the left and to the right can be

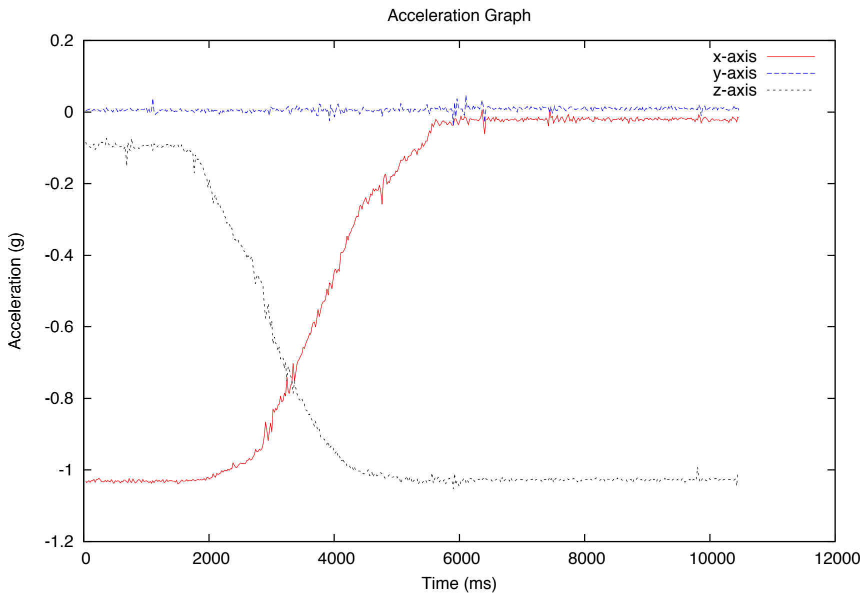


Figure 5.2: Acceleration graph recorded for rotating 90° around the y-axis. (Blue = x, Black = y, Green = z)

seen in Figure 5.3. The acceleration is measured in gravitational force (g-force) and the time in milliseconds, where $1g$ is the gravitational pull. The accelerometer for the chosen phone have a measurement limit between $-2g$ and $2g$.

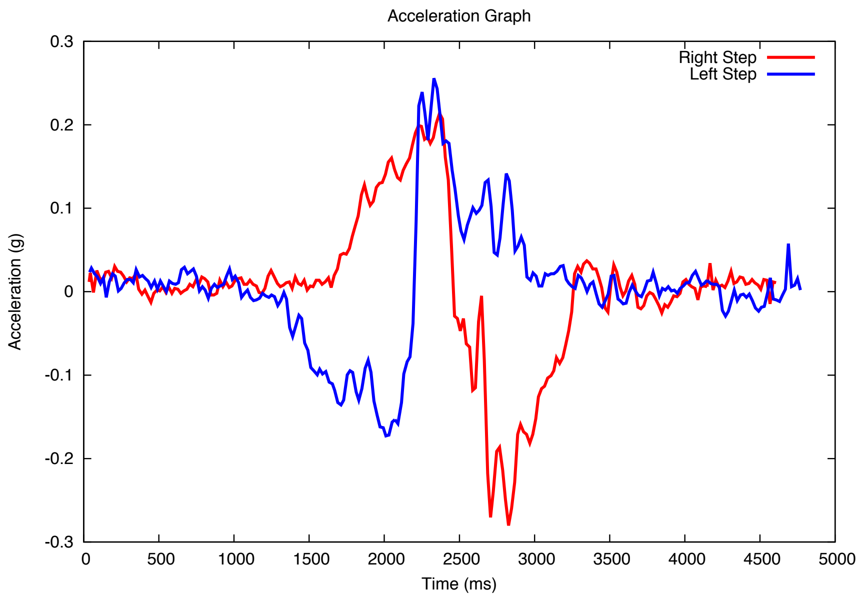


Figure 5.3: A graph showing a left, blue, and a right, red, step.

5.2 Gyroscope

The gyroscope measures the rotational velocity (rad/s), meaning how much the orientation of the phone has changed since last reading. The orientations of the phone is referred to as *Pitch*, *Roll*, and *Yaw* and is measured around the axes x, y, and z, respectively as illustrated in Figure 5.4. As an example, if the user rotates the device 20° around its x-axis, Pitch will, over the course of the action, accumulate to 0.35 radians, whereas Roll and Yaw remain unaffected.

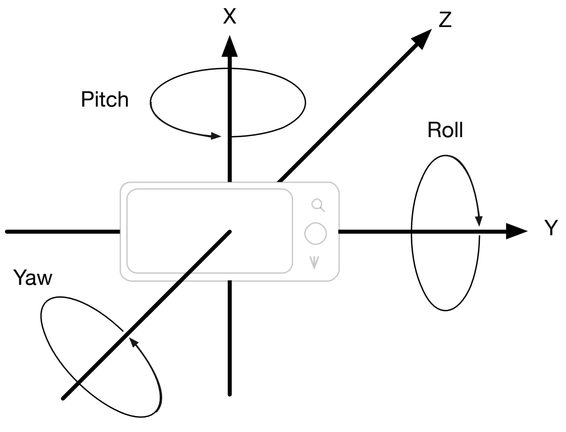


Figure 5.4: Pitch, Roll and Yaw illustrated.

The gyroscope can be used in the application to accommodate for tilting the phone while playing. Without this accommodation the gravitational pull would affect the acceleration-reading of the y-axis, when the phone has a yaw-rotation. Furthermore, if the phone has a pitch-rotation, the acceleration in the y-axis is smaller than intended, which can be corrected using the gyroscope and is further explained in Section 6.6.

When using the gyroscope, the accumulation of each gyroscope reading is needed to determine the correct angle that the phone is tilted. Due to noise in the gyroscope readings, the accumulated value will be inaccurate. The inaccuracy can cause drifting, which means the gyroscope cannot track when the system is back to its original position. To avoid drifting, the gyroscope can be utilised together with the accelerometer to correct the uncertainties of the gyroscope [33, 34].

To test how much the angle drifts, the device was positioned on a flat, undisturbed surface for 3 minutes. The recording can be seen in Figure 5.5 where one graph illustrates raw data received from the gyroscope, and another graph illustrates the same dataset run through a complementary filter.

As is clearly evident by the graphs, implementing a complementary filter would greatly increase the accuracy of the application. The sensor fusion is implemented using a complementary filter to have the acceleration reading adjust the accumulated angle of the gyroscope. The formula is as follows:

$$angle = \alpha * (angle + gyro * \Delta t) + (1 - \alpha)(angle_{acc})$$

where,

$angle$ is the accumulated angle.

α is the coefficient which determine how much each sensor affects the result.

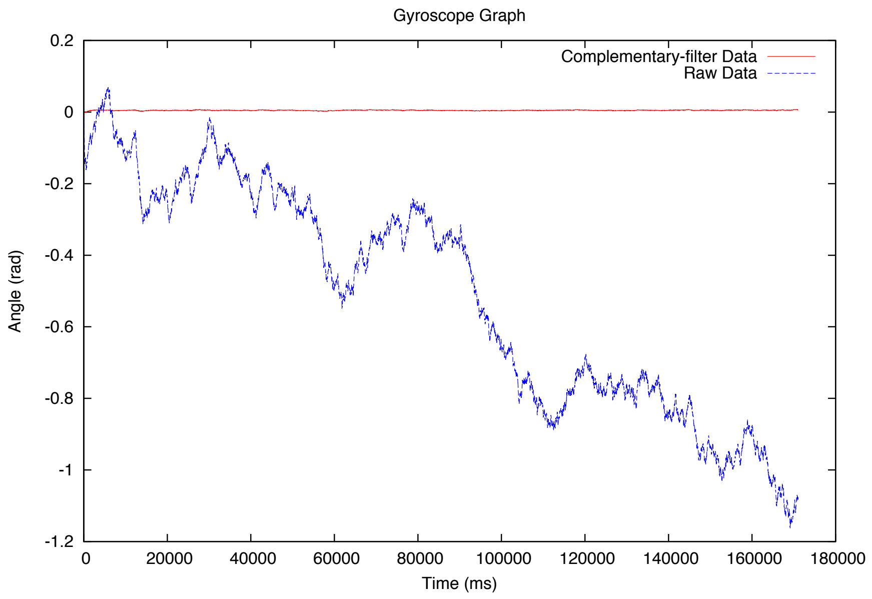


Figure 5.5: The complementary-filter over a 3 minute period.

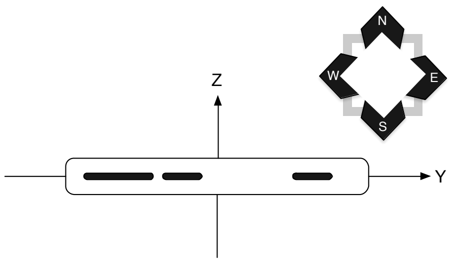
gyro is the gyroscope reading measured in rad/s.

$angle_{acc}$ is the angle, determined by the acceleration, measured in rad and found using atan2 function.

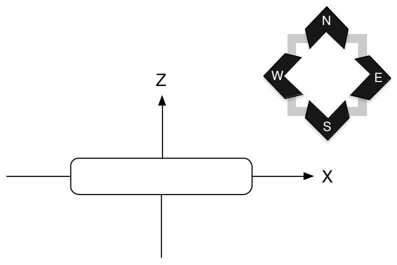
5.3 Magnetometer

The smartphone moves along the y-axis, but sometimes the phone rotates around the x- and z-axis when the player moves, which gives a smaller acceleration in the y-axis. To correct the accelerometer reading, the magnetometer can be used. Using the magnetometer, the initial orientation can be found, which then can be compared to the current orientation to find the angle difference. To find the orientation of the magnetometer, the phone was held in four different positions.

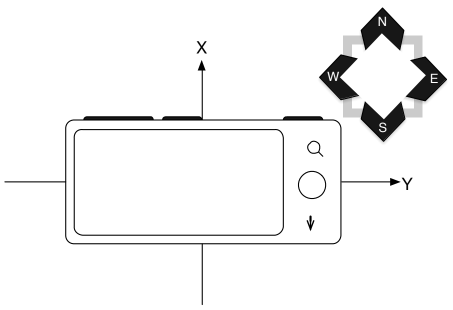
Drawings of these positions can be seen in Figure 5.6. Each drawing and its corresponding caption, shows the geographical north pole and the heading of the smartphone. Figure 5.6a illustrates how the phone should be held when the application is in use and the +z-axis is directed north. The magnetometer showed an angle of 270°, which is west, hereby, it could be concluded that the orientation of the magnetometer followed the -y-axis. When the phone is tilted as seen in Figure 5.6b and Figure 5.6d, the y-axis was now in a vertical direction and the +z-axis was directed north, the magnetometer gave an angle of 0° and 90○, respectively. This angle indicated that when the y-axis is at a vertical direction the +z-axis becomes the direction in which the magnetometer measures its angle. Figure 5.6c were used to confirm that whenever the y-axis is in a horizontal direction, the -y-axis is the direction in which the magnetometer measures its angle. When the different directions of the magnetometer are known, they can be used to correct the affected readings from the acceleration in the y-axis.



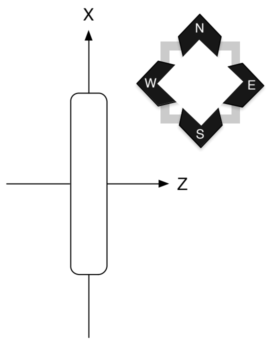
(a) The side of the phone.
Magnetometer reading: 270°.



(b) The top of the phone.
Magnetometer reading: 0°.



(c) The front of the phone, in landscape mode.
Magnetometer reading: 270°.



(d) The top of the phone.
Magnetometer reading: 90°.

Figure 5.6: The orientation of the phone and its corresponding magnetometer reading. All drawings are seen from above.

5.4 Data logger

All the data in this project has been logged, using a Nokia Lumia 820 running Windows Phone 8. The data logger has been used to understand the output of the phone's different sensors. It works by recording output from the accelerometer, the gyroscope, and the magnetometer, the collected data can then be sent to a web-service, which stores the data in a database. To make sure that the data logger does not overburden the database with update-requests, all the sensor-data is stored locally, and sent to the web-service once recording is complete. The data has been sent through a web-service because Windows Phone 8 does not offer a framework which works directly with MySQL.

6 Implementation

The implementation chapter contains a description of the different attempted dead reckoning approaches and the required constraints. Thereafter, an overview and implementation of different classes is shown. Finally, precision tests is analysed, in order to determine how accurate the implemented dead reckoning technique is.

6.1 Dead Reckoning Approaches

When implementing dead reckoning techniques, various approaches were attempted, which had different strengths and weaknesses. It proved to be a good learning experience, as it indicated what considerations were key to implement dead reckoning.

Initial Approach

The initial approach was primarily based on the theory in Section 4.1, which explained the relation between acceleration, velocity, and position. It was done with an algorithm that recorded all accelerations. The algorithm used these values to calculate the velocity and the user's estimated position. This approach was simple but suffered from noise, as the noise would also change the velocity and thus the position, without the user actually moving.

Filters

To reduce the noise impact on the position, a filter was implemented, see Section 4.2.2. It proved to be a fast and simple approach to reduce the noise impact. In order to prevent noise from having an impact on the position, when the user is not moving, an algorithm was developed. This approach hindered the algorithm from registering the position change, as it would only change the position once a complete step was taken.

Another hindrance for the algorithm was when taking many steps to one side. Which rendered the algorithm unable to determine whether a step was to the left or to the right, as the steps were done with too high frequency. For that reason, this algorithm was deemed undesirable as the intention of the algorithm was to be used in games, which requires a fast update. However, it would not have been a concern if updating the position during a step was not necessary.

Constraints

With the previous approaches in mind, another algorithm was made, which had the same general idea of calculating the position with the theory from Section 4.1. A calibration was done to filter out the acceleration offset noise.

Furthermore, when developing an algorithm with this approach, different constraints were added, to cancel out any undesired position-estimation updates. One constraint was setting a limit on the area a user was allowed to move within. In the case where the user was estimated to be beyond this area, the position was set to the edge of the area.

Another constraint was that if the acceleration was too low, it would be discarded. This acceleration was discarded since it would give a change in position when a step was not performed.

In addition, if the acceleration was under a specific limit for too long, the velocity was set to zero, as it was interpreted to the user standing still. Although this algorithm took noise into account, and the position accuracy was improved. A problem remained, if the phone was tilted as the gravitational pull would have an impact on the acceleration registered. Finally, whenever a step was taken, the deceleration would make the estimated position move backwards, since the deceleration was bigger than the corresponding acceleration, resulting in a velocity with opposite direction.

In order to take the gravitational pull into account, the *Motion* class from Microsoft [35] was tested. The class used the accelerometer in conjunction with the gyroscope and the magnetometer to cancel out the gravitational pull. This seemed as an easy way to accomplish the goal. However, it was unclear how the Motion class altered the data, and for that reason it was not used further.

Bayesian Structure

With these approaches in mind, a dynamic Bayesian network was modelled. It proved to give a good overview of the different calculations needed to determine the user's position. In addition, the marginal probability distribution for each stochastic variable was found by use of the theory from Section 4.4. The variance, found when determining the marginal probability distribution, was not used, since no observations were used. However, by calculating the variance, with observations on the position nodes, the network would be an extended Kalman filter. The variance would then have an influence on the mean of the marginal probability distribution of the position, as described in Section 4.4.4.

Corrected Moving Average

A problem found was the velocity gain, because of the deceleration being larger than the acceleration, resulting in the velocity being in the opposite direction than what was performed. This problem was solved by adjusting the deceleration to be the same as the acceleration with opposite direction. The adjustment was a hack in the Bayesian network, as the mean value for the deceleration nodes were assigned to be the same but in the opposite direction of the accelerations of the step. The adjustment is different from the direct combination approach, as described in Section 4.4.3, and therefore it is a hack. By implementing this hack, the velocity gain problem was corrected, but resulted in a limitation of how fast a user could take steps. It was deemed to not being a problem, as the game became playable, with the constraint in mind.

Rotation

With the correction of the deceleration problem, another problem still existed, which was accidental phone-rotation when moving. To correct for pitch-rotation, the magnetometer of the phone was first used to determine the angle of the pitch-rotation. This angle would then be used in conjunction with trigonometry to correct the acceleration when moving.

The yaw-rotation had to be taken into account, as the gravitational pull would affect the measured acceleration. In order to correct for gravitational pull, the gyroscope was used in conjunction with the accelerometer to determine the yaw-rotation angle. The gyroscope would, over time, drift and make the angle inaccurate. To avoid drifting, a complementary filter was implemented. The complementary filter used the angles determined by accumulating the gyroscope readings, as well as determining the angle by use of the accelerometer readings.

With the correction of the gravitation pull implemented, it was found that the gyroscope and accelerometer approach was better than using the compass for this application. Therefore, the gyroscope was used to handle both pitch- and yaw-rotation, as the compass was slow to adjust to a new angle, whereas the gyroscope and accelerometer was faster.

After having used these various approaches, four constraints remained. These constraints will be explained in more details in the next section.

6.2 Constraints

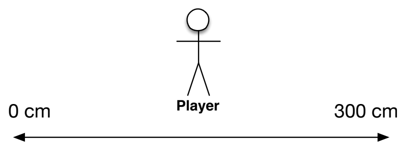
The purpose of this section is to describe a set of constraints. The reason for doing so, is the limited time allotted for the project. The constraints that are applied to the project have each been depicted in Figure 6.1. Each constraint is explained below, the letter in the list corresponds to the letter in Figure 6.1.

(a) To simplify the complexity of the application, the movement-area for the player will be a fixed size of $300cm$ in width. It means that the player will have to play within the movement-area.

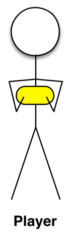
(b) To ensure consistency in accelerometer readings, a player must hold the device in landscape mode, with the touch pad buttons to the right and the x-axis in a vertical line. As it is impossible to keep the phone completely steady while moving, the application will allow for a rotation of up to 30° around the x- and z-axis.

(c) The maximum velocity is limited to $3m/s$, as the player should not be able to move faster than this.

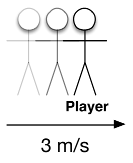
(d) Acceleration is constrained to $1g$. From observations, accelerations with a value exceeding $1g$ does not happen while side stepping, this includes both negative and positive acceleration values.



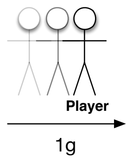
(a) Fixed movement area.



(b) Fixed device position.



(c) Maximum velocity.



(d) Maximum acceleration.

Figure 6.1: Constraints

6.3 Class diagram

In this section, the class diagram of the dynamic Bayesian network, see Figure 6.2, will be shown and explained, to give an overview of the structure of the Bayesian network implementation.

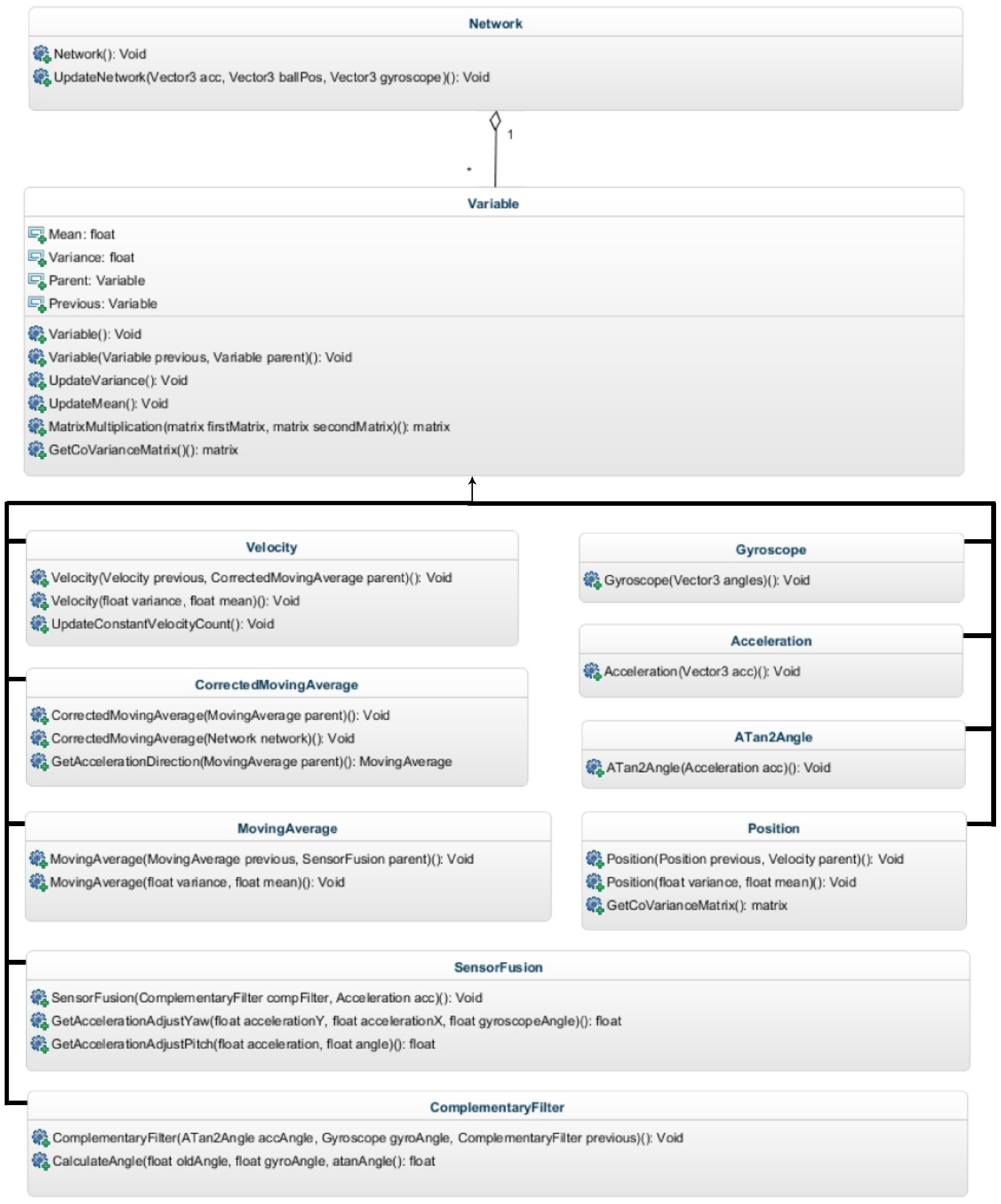

Network
Network(): Void
UpdateNetwork(Vector3 acc, Vector3 ballPos, Vector3 gyroscope)(): Void
1
*
Variable
Mean: float
Variance: float
Parent: Variable
Previous: Variable
Variable(): Void
Variable(Variable previous, Variable parent)(): Void
UpdateVariance(): Void
UpdateMean(): Void
MatrixMultiplication(matrix firstMatrix, matrix secondMatrix)(): matrix
GetCoVarianceMatrix()(): matrix
Velocity
Velocity(Velocity previous, CorrectedMovingAverage parent)(): Void
Velocity(float variance, float mean)(): Void
UpdateConstantVelocityCount(): Void
Gyroscope
Gyroscope(Vector3 angles)(): Void
Acceleration
Acceleration(Vector3 acc)(): Void
CorrectedMovingAverage
CorrectedMovingAverage(MovingAverage parent)(): Void
CorrectedMovingAverage(Network network)(): Void
GetAccelerationDirection(MovingAverage parent)(): MovingAverage
ATan2Angle
ATan2Angle(Acceleration acc)(): Void
MovingAverage
MovingAverage(MovingAverage previous, SensorFusion parent)(): Void
MovingAverage(float variance, float mean)(): Void
Position
Position(Position previous, Velocity parent)(): Void
Position(float variance, float mean)(): Void
GetCoVarianceMatrix(): matrix
SensorFusion
SensorFusion(ComplementaryFilter compFilter, Acceleration acc)(): Void
GetAccelerationAdjustYaw(float accelerationY, float accelerationX, float gyroscopeAngle)(): float
GetAccelerationAdjustPitch(float acceleration, float angle)(): float
ComplementaryFilter
ComplementaryFilter(ATan2Angle accAngle, Gyroscope gyroAngle, ComplementaryFilter previous)(): Void
CalculateAngle(float oldAngle, float gyroAngle, atanAngle(): float

Figure 6.2: Class diagram for developed navigation library.

To implement a dynamic Bayesian network, the structure from Figure 6.2 was used. The *Network* class keeps track of the overall structure of the dynamic Bayesian network. In order to keep track of the structure, the *Variable* class has nine subclasses. The Variable class exists to calculate the marginal probability distribution for each of its subclasses. Each subclass then calculates the concrete probability distribution and constraints, which varies for each of the subclasses.

Concrete implementation of a sample of the classes in Figure 6.2 will be described in the next sections.

6.4 Network

The Network class is implemented to keep track of the current timeslice of the dynamic Bayesian network. The Network contains nine properties corresponding to the nodes for a given timeslice. A timeslice can be seen in Figure 4.6.

```
public Network()
{
        Acceleration = new Acceleration(Vector3.Zero);
        Gyroscope = new Gyroscope(Vector3.Zero);
        ATan2 = new ATan2Angle(Acceleration);
        ComplementaryFilter = new ComplementaryFilter();
        SensorFusion = new SensorFusion(ComplementaryFilter, Acceleration);
        MovingAverage = new MovingAverage(0.0f, 0.0f);
        CorrectedMovingAverage = new CorrectedMovingAverage();
        Velocity = new Velocity(0.0f, 0.0f);
        Position = new Position(0.0f, Constants.POSITION_START);
}
```

List 6.1: Constructor for Network

When the Network is initialised, all the variables of the Network get assigned to their default value as seen in List 6.1. The reason for setting the variables to their default values, when initialising the Network, is to have a fixed-point to update the next timeslice. After the Network has been initialised, a similar method can then be used to update the network, for a new timeslice, corresponding to a new accelerometer reading.

```
public void UpdateNetwork(Vector3 newAcceleration, Vector3 newGyroscope)
{
        Acceleration = new Acceleration(newAcceleration);
        Gyroscope = new Gyroscope(newGyroscope);
        ATan2 = new ATan2Angle(Acceleration);
        ComplementaryFilter = new ComplementaryFilter(ATan2, Gyroscope,
            ComplementaryFilter);
        SensorFusion = new SensorFusion(ComplementaryFilter, Acceleration);
        MovingAverage = new MovingAverage(MovingAverage, SensorFusion);
        CorrectedMovingAverage = new CorrectedMovingAverage(MovingAverage);
        Velocity = new Velocity(Velocity, CorrectedMovingAverage);
        Position = new Position(Position, Velocity);
}
```

List 6.2: UpdateNetwork method in Network

List 6.2 shows the method for updating the network to a new timeslice. The `UpdateNetwork` method updates the probabilities in the order of the dynamic Bayesian network, from parents to children, where each probability distribution update is handled in the given classes that are instantiated. The *Acceleration* class, which is part of the Network, will be described hereafter.

6.5 Acceleration

Acceleration is a class implemented to represent the *Acc* node from the Bayesian network. It inherits from the abstract Variable class, where the Variable class serves as a structure that all nodes in the network must have.

Since Acceleration inherits from Variable it has to implement two methods, `UpdateVariance` and `UpdateMean`. The mean value is the accelerometer reading. The variance for the acceleration is a constant value and is determined by testing, which is described in Section 6.5.1. Furthermore, when inheriting from Variable a matrix multiplication method is provided, as well as four properties, keeping track of a stochastic variable's mean, variance, and its parents.

The readings from the accelerometer were uncertain, as of such, the variance for the accelerometer readings was determined, which is described hereafter.

6.5.1 Finding the Variance

To measure the distribution of noise from the accelerometer, the formula for finding variance, as shown in Section 4.4, will be utilised. The device was placed on a flat surface for a period of three minutes, to record a set of accelerations containing noise. This data was used to calculate μ, which is the mean value for the stochastic variable X. The expected value would be zero, as the device was not in motion when the data was recorded. The following formula was used to approximate the mean acceleration:

$$\mu = \frac{1}{n}\sum_{i=1}^{n}(X_i)$$

where, n is the amount of data entries and X_i is the i'th entry. For the chosen device, the mean value was calculated to **0.00026 g**. When the mean value is known, it can be used to approximate σ^2, which is the variance which can be approximated as follows:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2$$

where, n is the amount of data entries, X_i is the i'th entry, and μ is the mean value. For the chosen device, the variance was calculated to **0.00552 g**.

By finding the mean value and the variance, a normal distribution can be implemented on measured accelerations by using the Equation (4.2) as repeated below:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2*\sigma^2}}$$

Figure 6.3 shows the final result of finding the normal distribution. The red points plotted in the graph, is the amount of times the specific value has been recorded during the three minutes of recording. The blue graph shows the normal distribution which has been calculated from the recorded values. It can be seen that the red points and blue graph are not aligned. However, it is not an issue, since the ordinate axis specifies amount and density for the red points and blue graph respectively.

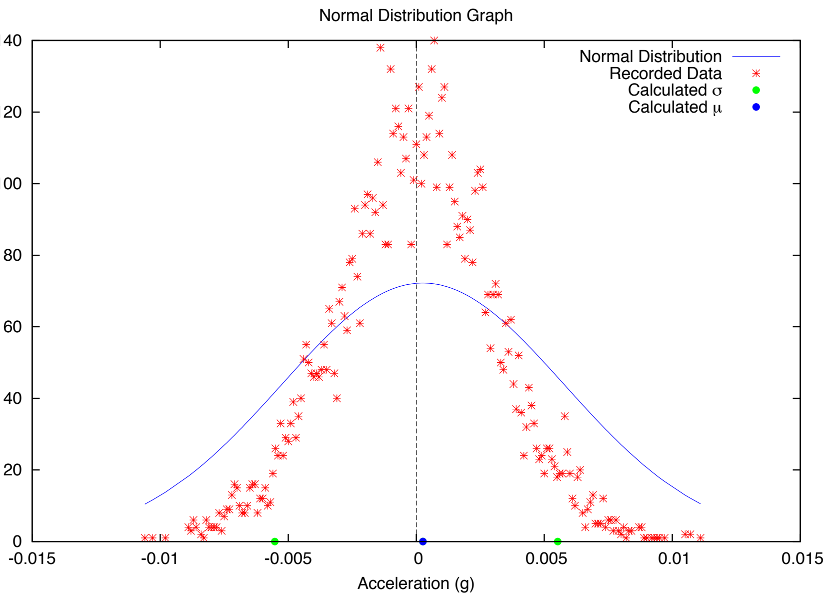


Figure 6.3: Finding μ, σ, and the normal distribution

6.6 SensorFusion

The *SensorFusion* class is implemented to adjust for yaw and pitch-rotations, where the yaw and pitch-rotation angles are provided in the constructor. The SensorFusion adjusts the acceleration readings in the y-axis in two methods.

The method in List 6.3 corrects the acceleration when the phone is tilted around the z-axis, and is important since it is affected by the gravitational pull.

When the yaw-rotation has been found, using the accelerometer and gyroscope, the gravitational pull is taken into account. The method, seen in List 6.3, calculates movement acceleration, without gravitational pull, is found using trigonometry.

```
private float GetAccelerationAdjustYaw(float accelerationY, float
    accelerationX, float gyroscopeAngle)
{
        return (float)((accelerationY - (-accelerationX *
            Math.Sin(gyroscopeAngle))) / (Math.Cos(gyroscopeAngle)));
}
```

List 6.3: Method correcting acceleration when tilted around the z-axis.

An illustration of the gravitational pull problem can be seen in Figure 6.4, which is used to explain List 6.3.

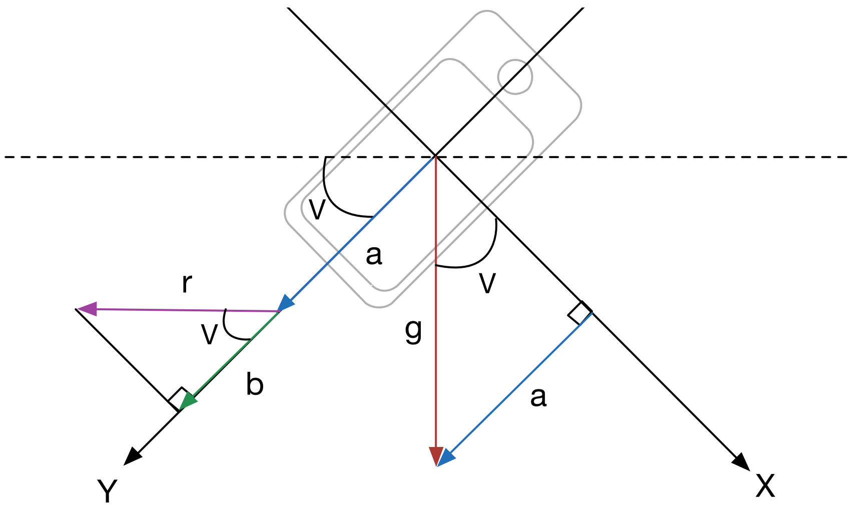


Figure 6.4: Yaw-rotation adjustment.

In Figure 6.4, the phone is tilted in an angle of V. r is the desired acceleration, as it is the acceleration, in the y-axis, without gravitational pull. $a + b$, is the measured acceleration in the y-axis of the accelerometer. g is the gravitational pull, which is constant. g is used in combination with the angle V to calculate a, which is how much the gravitational pull affects the measured acceleration in the y-axis.

$$a = g \cdot sin(V) \tag{6.1}$$

Once a has been found, using Equation (6.1), b can be found by subtracting a from the measured acceleration in the y-axis. b is the acceleration in the y-axis, unaffected by gravitational pull, however, the desired acceleration is in the horizontal plane. As of such, V and b, is used to find r, seen in Equation (6.2).

$$r = \frac{b}{cos(V)} \tag{6.2}$$

The measured acceleration in the y-axis, m_{acc}, is given by $a + b$. Hence, the final formula for finding r is Equation (6.3).

$$r = \frac{m_{acc} - g \cdot sin(V)}{cos(V)} \tag{6.3}$$

Much like the adjustment in the yaw-rotation, a similar correction is performed for pitch-rotation. However, for pitch-rotation there is no gravitational pull affecting the measured acceleration. For that reason, it is sufficient to use Equation (6.2), as b is the measured acceleration. This correction has to be calculated after the correction of the yaw-rotation.

The calculation of the marginal probability distribution of the SensorFusion node is not done with a direct combination. For that reason, the variance can not be determined as the marginal distribution of the nodes that are constructed with a direct combination. Although, the current implementation of calculating the variance of the SensorFusion node as the sum of the variance of the parents is not correct, it is done since a better solution was not present.

```
private float GetAccelerationAdjustPitch(float acceleration, float angle)
{
        return acceleration / (float)Math.Cos(angle);
}
```

List 6.4: Method correcting acceleration when tilted around the x-axis.

6.7 CorrectedMovingAverage

The *CorrectedMovingAverage* class adjusts the acceleration, such that a deceleration of a step is the same size as the acceleration of the step. This adjustment is done with the help of a stack which saves each step of an acceleration, such that it can be used for the corresponding deceleration. As mentioned in Section 6.1, it is a hack, since it does not follow the defined rules for updating the marginal probability distributions. However, it proved to be a good solution that takes the velocity gain into account.

To track the direction of a step, an enumerator was created which had the elements *Unknown*, *Left*, and *Right*. When the step direction is unknown, a step has not been detected yet and the program checks each acceleration to see if it is outside the jitter zone. Jitter noise occurs when the user unintentionally shakes the phone. The jitter zone is an interval specifying where jitter noise can occur. When the acceleration is higher than the jitter zone, a right step has begun, and if it is lower, it is a left step. This categorisation is done with the knowledge of how an acceleration graph for a step is shaped, and is found in Section 5.1. In addition, when a step has been detected, the velocity is reset to zero as the step has just begun.

If a step is in progress, every next acceleration, which is in the same direction, is loaded into a stack. Once the deceleration has begun, the stack is popped for each consecutive deceleration.

6.8 Position

The *Position* class is used to track the position of the phone, and is a subclass of the Variable class. The position has three parents, which is the previous position, current velocity, and the ball position corresponding to the dynamic Bayesian network described in Section 4.3.1.

In List 6.5, the `UpdateMean` method has been implemented to take all parents into account for the Mean, as can be seen on line 3. However, a constraint on the mean of the position has also been implemented, which is the limit of the area, corresponding to walking into the wall, see Figure 6.1a and can be seen in line 5. The constraint resets the variance of the position in line 12, and resets the mean and variance of the velocity in line 13-14, and is implemented to handle uncertainties in the positioning measurement.

6.9 Precision Test

Once the application was developed, the datalogger was used to gather new data such that a precision test could be performed. Data was recorded in distances of 1, 2, 25, and 50 meters. These data were used to determine the correctness of the output given by the application.

Tables showing the test results can be seen in Appendix B. The majority of the tests showed that the application underestimated the calculated positions and the average calculated position was 46.1% of the actual position. The percent deviation ranged from 22.8% to 164.1%, however, only 6 calculated positions were above 70% while 81 were below 70%. To summarize the percent deviation, a bar chart was made to show an overview of the data, see Figure 6.5.

```
protected override void UpdateMean()
{
    Mean = Constants.POSITION_MEAN_GAIN + Constants.F_POSITION[0, 0] *
        Parent.Mean + Constants.F_POSITION[0, 1] * Previous.Mean;

    if (Mean > Constants.POSITION_MAX || Mean < Constants.POSITION_MIN)
    {
        if (Mean > Constants.POSITION_MAX)
            Mean = Constants.POSITION_MAX;
        else
            Mean = Constants.POSITION_MIN;

        Variance = 0.0f;
        Parent.Mean = 0.0f;
        Parent.Variance = 0.0f;
    }
}
```

List 6.5: Update mean method containing constraints for position

Various reasons for this underestimation could be the filters suppressing the accelerations, the rotation of the phone still having affect on the accelerations, or that the sensor readings in general measure too low values.

This underestimation could be corrected in various ways. One way could be to scale the position with a factor, or to make further adjustments in the actual implementation. Some of these adjustments are discussed in Chapter 7.

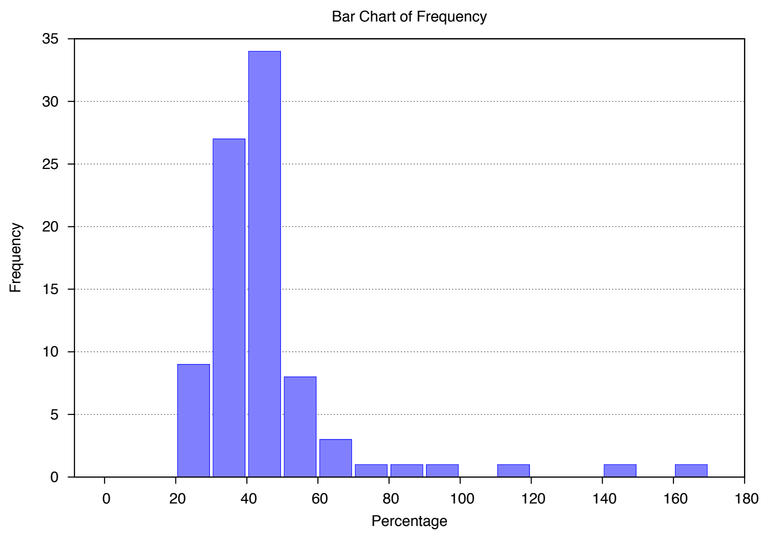


Figure 6.5: A bar chart for the percent deviation range and how many times it has been observed.

7 Discussion

While developing the application for this project, multiple problems were found, resulting in inaccurate dead reckoning positioning. Although many of the problems were solved, some problems still exist, which causes the dead reckoning to be inaccurate. Possible solutions to the existing problems in the application will be examined and discussed in further development, Section 7.2.

Apart from the existing problems, implementation decisions, made during development of this application, are discussed in this chapter. Furthermore, the application will be compared to existing systems to determine the pros and cons of this application.

Platform

The application that is developed for this project is currently working on Windows Phone 8 platform. The choice of developing for Windows Phone 8 gives the opportunity to distribute to a large market, where Windows Phone 8 had 4.2 million users in 2012 [36]. The market could, however, be expanded if the application was to be developed for other platforms as well, but the Windows Phone 8 platform still allows for a large market. However, with the amount of time at hand the resources were allocated to develop a dead reckoning application for a single platform, as the focus was to make dead reckoning work, rather than distributing the application for multiple platforms.

Furthermore, the variance for the acceleration is determined for one accelerometer and thus the variance is not directly portable to other phones, which may have other variances for the acceleration even if they have the same accelerometer model. For that reason, it could be argued that when the application is first used, the phone should be placed on a flat surface for a given time period to determine the variance of the acceleration for that specific accelerometer.

One Dimension

For the scope of this project, dead reckoning has been implemented for one dimension. It was done to simplify how movements could look, thus making dead reckoning easier to implement.

A constraint was that the user should do sideways movements. However, this constraint was not a problem since the application only needed one dimensional movements and there are a lot of arcade games that just need one dimensional movements.

The movement tracking could be in two dimensions for a planar surface, or even in three dimensions when there is a desire to track vertical movement as well. Movement tracking in two dimensions, could be possible by keeping track of the phone's heading.

Three dimensions would make tracking more complex, since it would require something to track the movement in three axis. However, with more time, tracking jumps could potentially be pursued as it would not require additional environmental constraints, other than a place where it is possible to jump freely.

Pedestrian Dead Reckoning

The application developed for this project was an algorithm using pedestrian dead reckoning to track the user's movements. It gave certain opportunities, while removing others. The fact that pedestrian dead reckoning uses the sensors of the phone itself, with no dependencies from other sources, could be used to track the user's movement.

Pedestrian dead reckoning gave the advantage that extra equipment, such as a Wi-Fi beacon, was not needed, and was the primary reasoning for using pedestrian dead reckoning. However, it also limited the scope of what information could be used to determine the movement of the user. A problem when not using such information, was that after some time, the pedestrian dead reckoning became highly inaccurate and the variance would be higher. This would result in a higher uncertainty. However, implementing position tracking by another method could be used as evidence on the Bayesian network. This other method could be implemented in conjunction with the pedestrian dead reckoning as an extended Kalman filter to increase the accuracy of dead reckoning.

Motivation

Various features were implemented in the game developed, to motivate people to play the game. These features included a high score, power-ups, and increasing difficulty.

The high score simply shows the highest scores achieved on the device. Furthermore, the high score is intended to make the user compete against other users, thereby making people want to play more to improved their score or to be better than their friends.

To make the game entertaining, power ups and increasing ball speed were implemented, since it was believed that without those features the game would become tedious and thus less entertaining.

As mentioned above, features have been implemented to make the game entertaining and at the same time make people exercise. Whether the features make the game entertaining and motivates the users towards exercise, has not been examined. In order to examine this, the target audience of the application would have to be involved to test the game and give feedback on what they find entertaining. Furthermore, a fitness consultant could be contacted to determine whether the movement patterns needed to play the game is good exercise or not.

7.1 Perspective

With the dead reckoning exercise application developed for this project, the application can be compared to other existing exercise systems that is described in Section 2.2.

The game that has been developed is for exercise purposes. It is believed that Endomondo and Zombies, Run! have the potential to evoke exercise on a more intensive level, due to the nature of running versus sidestepping. However, Moves does not in itself lead to exercise as the developed application, but tracks the progress of a whole day and not only when the user wants to exercise.

When using applications that induce or track exercise, the user should gain some value from it. For this project the primary focus was entertaining exercise, while systems such as Endomondo and Moves gives the value of tracking the exercise performed. A system that has been examined in this project and gives entertainment value is Zombies, Run!. Zombies, Run! brings an interesting storyline, written by professional authors and performed by professional narrators, while running. Furthermore, it entertains the user while exercising, however, if a user does not like the story, the primary entertainment value of the Zombies, Run! is counter productive, as the user would then stop playing due to him perceiving that it is a bad story. The developed application is for the user to entertain himself by playing the game and achieving a better high score. However, therein lies the problem that there is no storyline to make the user motivated to play the game, and thus the game may become a tedious experience.

When doing exercise, information about progress, such as, calories burned or the amount of exercise performed can be presented. The application was developed to make the user have fun when doing exercise, and thereby motivate people, after having used the application, to begin doing other more intensive types of exercise afterwards. Therefore, information about calories burned and the amount of exercise performed was not an important part, as it is for Endomondo, Moves, and Zombies, Run!. Implementing tracking of the amount of exercise performed when using the application could be an additional feature that motivates people. However, due to the primary focus being dead reckoning and embedded systems, exercise statistics have not been a concern.

As discussed, the application developed differs in various ways from the existing systems described in Section 2.2, and has had a different approach to exercise. The developed application is deemed to be different than the other systems. It serves a different purpose where the user plays with his movement and exercise comes as a bonus, in contrast to the other systems, where exercise has a more central and visible role. In that regard, the developed application is not unique when it comes to the game itself, but it is the movement tracking by dead reckoning, working as the controller, which makes the developed application interesting.

7.2 Further Development

During this project, a number of problems have been encountered. These problems were discussed in Chapter 7, ideas and features to solve these problems will be presented in this section.

7.2.1 Cross platform

As discussed earlier in this chapter, this project was focused on Windows Phone 8 using a Nokia Lumia 820. However, making the application available on multiple platforms would open up to a bigger market. To do this effectively, code could be ported to Unity which was shortly described in Section 3.2.1. Unity works with C# which would make it easy to port the code, as it is able to compile the code for Windows Phone, iPhone, and Android. Next, a solution to the problem of noise on individual phones must be found. This problem could be solved by calibrating when the application is used for the first time. The calibration would give the application all the information it needs to run correctly on the individual phone.

7.2.2 Rotation Matrix

The application corrects the tilting of the phone when a yaw-rotation or pitch-rotation is done by the user. However, when the phone is rotated in both pitch and yaw, the correction of the acceleration is inaccurate. This inaccuracy happens because the acceleration has been used to determine the angle of the pitch and yaw before the acceleration in itself is corrected.

To correct the acceleration when the phone is tilted in pitch, yaw, and roll, a rotation matrix can be used to multiply on the gravitational pull vector, $\vec{G} = \begin{bmatrix} 0 & 0 & g \end{bmatrix}$ [37].

The implementation of the rotation matrix to correct for tilting, has not been done due to time constraints. However, if it was to be implemented, it would replace the SF node in the dynamic Bayesian network, the atan2 node, and the Comp node would have to find the missing angles.

In addition, if the rotation matrix was implemented, it could adjust the gravitational pull in three dimension and thereby simplify the change to additional dimensions.

7.2.3 Additional Dimensions

This project investigated dead reckoning in one dimension, but it is possible to expand the application to two or three dimensions. Getting dead reckoning to work in two and three dimensions would enable a wide array of new gameplay and, hereby, the variation of exercise. Considering two dimensions, it is possible to move forwards, backwards, sideways, and diagonal. Since the precision of dead reckoning in one dimension is inaccurate, it is assumed that it is also inaccurate in two dimensions. To solve the inaccuracy of dead reckoning, observations such as Wi-Fi fingerprinting could be implemented. Although, when moving in three dimensions, additional problems can arise.

7.2.4 Motivational Factors

To implement a feature where players could compete against their friends, social networks such as *Facebook* could be integrated. Social networks would offer a platform from which social relations have already been established and thus simplify the process of the development of the application. Being able to turn the social network feature on or off would prevent the issue with people being discouraged from playing the game in case they want their game progress to be private. Other factors could be developing additional games, such that people had a variety of games to play. Furthermore, motivational features can always be strived for to make the user exercise more, as discussed in Section 2.9.

7.2.5 Machine Learning

Machine learning is a technique in which the system saves data and uses it to create or enhance a model. Some factors in the application are unknown and could be found using machine learning to help improve dead reckoning. An example of these factors could be the variance gain from Section 4.4, which is currently a fixed estimated value but could vary for different phones. Machine learning could be taxing for the phone and cause new problems, such as additional technology dependencies, excessive storage usage, and system delay.

8 Conclusion

To conclude this project, each of the three questions have to be answered to confirm or invalidate the hypothesis.

The first question was:

What are the requirements for a smartphone application that make people want to exercise?

It was found that the requirements were primarily *Exercise*, *Dead Reckoning*, and *Game*. All of these requirements were implemented in the application, in the form of a game using dead reckoning to track the user's movement. The movement was used as a controller for the game, which made the user exercise. It is believed that the game would make people want to exercise by playing the game, as using movement to play games have proven to be fun for other applications, such as the *Nintendo Wii* and *Xbox Kinect*. Furthermore, the movement tracking as controller can easily be used in a wide range of other applications, such as various old arcade games. As of such, there is a lot of options to use the movement tracking for a given user that may like one game and not another.

Whether the sidestepping movement is considered exercise is up for discussion. However, if exercise is defined as any form of body movement, it is considered exercise.

The second question was:

How can a smartphone application motivate people to exercise?

A smartphone application has been developed that may motivate people to exercise. Game elements, such as a high score, power-ups, and increasing difficulty, have been implemented, which were assumed to make the application fun and thus motivating for people. Furthermore, these elements were implemented to give some replay value, such that people would continue to play the game. Whether an entertaining game motivates people to exercise or not is debatable and subjective, see Section 7.2.4.

In addition, the game can be played everywhere and does not require information from external hardware, such as Wi-Fi and GPS, as it works if the user has a smartphone with an accelerometer and gyroscope.

The fact that the application utilises the user movement as the controller can make for a fun and engaging experience, which in turn can be a motivational factor to continue playing the game.

The third question was:

Why would people use this application instead of other applications?

As an intensive exercise application the user should not use this application over other applications, such as Endomondo and Zombies, Run!. However, it is believed with the current state of the application that it serves as a fun and engaging experience that may serve as a foundation for more intensive exercise. Furthermore, the application serves as an alternative to other applications on the market. The application developed focuses on the game as a way to exercise, whereas other applications make exercise the central part of the application.

Finally, the hypothesis was:

A game that is fun and motivates the user to exercise can be developed for a smartphone using dead reckoning.

A game has been developed for a smartphone that uses dead reckoning. Furthermore, the dead reckoning works, but some adjustments are needed to make the tracking more accurate. The question that remains is, if the game developed is fun and motivates the user to exercise. It is presumed that the game is fun and motivating for some people, and would be a good basis for the application, but this assumption would have to be examined.

Bibliography

[1] Louise Chang. Weight loss: Health risks associated with obesity, April 2012. http://www.webmd.com/cholesterol-management/obesity-health-risks.

[2] Morten Jastrup. Fedme koster os 14 milliarder årligt, September 2007. http://politiken.dk/mad/madnyt/ECE385273/fedme-koster-os-14-milliarder-aarligt/.

[3] Lung National Heart and Blood Institute. What causes overweight and obesity?, 2012. http://www.nhlbi.nih.gov/health/health-topics/topics/obe/causes.html.

[4] Mark A Pereira, Alex I Kartashov, Cara B Ebbeling, Linda Van Horn, Martha L Slattery, David R Jacobs Jr, and David S Ludwig. Fast-food habits, weight gain, and insulin resistance (the cardia study): 15-year prospective analysis. The Lancet, 365:36–42, January 2005. URL http://health-equity.pitt.edu/470/.

[5] Statens Institut For Folkesundhed. Overvægt og undervægt, 2010. http://www.si-folkesundhed.dk/upload/susy_2010_5_6_overv%C3%A6gt_og_underv%C3%A6gt.pdf.

[6] Jeffrey Yim and T. C. Nicholas Graham. Using games to increase exercise motivation. In Proceedings of the 2007 conference on Future Play, Future Play '07, pages 166–173, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-943-2. doi: 10.1145/1328202.1328232. URL http://doi.acm.org/10.1145/1328202.1328232.

[7] Helena. How many people use smartphones in the world, October 2012. http://www.onbile.com/info/how-many-people-use-smartphones-in-the-world/.

[8] Endomondo. Endomondo website. http://www.endomondo.com/about.

[9] Six to Start. Zombies, Run! http://www.zombiesrungame.com/.

[10] ProtoGeo. Moves. https://moves-app.com/.

[11] Michael Barr and Anthony Massa. Programming Embedded Systems: With C and GNU Development Tools. Number ISBN-10: 0-596-00983-6. O'Reilly Media Inc., 2nd edition, 1999.

[12] Navigation National Coordination Office for Space-Based Positioning and Timing. Official u.s. government information about the global positioning system (gps) and related topics. http://www.gps.gov/systems/gps/.

[13] Jesse Aronson. Dead reckoning: Latency hiding for networked games. Gamasutra, 1997. http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php.

[14] Stephane Beauregard and Harald Haas. Pedestrian dead reckoning: A basis for personal positioning. Proceedings of the 3rd workshop on positioning, navigation and communication, 1:10, 2006. http://www.wpnc.net/fileadmin/WPNC06/Proceedings/33_Pedestrian_Dead_Reckoning.pdf.

[15] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In Proceedings of the 2012 ACM Conference on Ubiquitous Computing, UbiComp '12, pages 421–430, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1224-0. doi: 10.1145/2370216.2370280. URL http://doi.acm.org/10.1145/2370216.2370280.

[16] Rommanee Jirawimut, Piotr Ptasinski, Vanja Garaj, Franjo Cecelja, and Wamadeva Balachandran. A method for dead reckoning parameter correction in pedestrian navigation system. IEEE transactions on instrumentation and measurement, 52 number 1:7, 2003.

[17] Lei Fang, Panos J. Antsaklis, Luis Antonio Montestruque, M. Brett McMickell, Michael D. Lemmon, Yashan Sun, Hui Fang, Ioannis Koutroulis, Martin Haenggi, Min Xie 0005, and Xiaojuan Xie. Design of a wireless assisted pedestrian dead reckoning system - the navmote experience. IEEE T. Instrumentation and Measurement, 54(6):2342–2358, 2005. URL http://dblp.uni-trier.de/db/journals/tim/tim54.html#FangAMMLSFKHXX05.

[18] Yoji Miyazaki and Toshiyuki Kamiya. Pedestrian navigation system for mobile phones using panoramic landscape images. In SAINT, pages 102–108. IEEE Computer Society, 2006. ISBN 0-7695-2508-3. URL http://dblp.uni-trier.de/db/conf/saint/saint2006.html#MiyazakiK06.

[19] Encyclopædia Britannica. dead reckoning. http://www.britannica.com/EBchecked/topic/154249/dead-reckoning.

[20] Robin Henniges. Current approches of wifi positioning. Technical report, TU-Berlin, 2012. http://www.snet.tu-berlin.de/fileadmin/fg220/courses/WS1112/snet-project/wifi-positioning_henniges.pdf.

[21] Arbitron Edison Research. The smartphone consumer 2012, 2012. http://www.slideshare.net/webby2001/the-smartphone-consumer-2012.

[22] Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. Object Oriented Analysis & Design. Marko Publishing ApS, Aalborg, Denmark, 1 edition, 2000.

[23] Xamarin. Compatibility, . http://docs.unity3d.com/410/Documentation/ScriptReference/MonoCompatibility.html.

[24] Xamarin. Release notes mono 2.8, . http://www.mono-project.com/Release_Notes_Mono_2.8.

[25] Microsoft Corporation. Xna game studio 4.0 refresh, 10 2013. http://msdn.microsoft.com/en-us/library/bb200104.aspx.

[26] Unity Technologies. Unity manual, 10 2013. http://docs.unity3d.com/Documentation/Manual/Input.html.

[27] Microsoft Corporation. Retrieving accelerometer input (windows phone). Website, 10 2013. http://msdn.microsoft.com/en-us/library/ff604984.aspx.

[28] Scott Miller, Kenneth Foust, and Izudin Cemer. Noise measurement, 2011. http://www.sensorsmag.com/sensors/acceleration-vibration/noise-measurement-8166.

[29] Steffen L. Lauritzen and Frank Jensen. Stable local computation with conditional gaussian distributions. Statistics and Computing, 11(2):191–203, April 2001. ISSN 0960-3174. doi: 10.1023/A:1008935617754. URL http://dx.doi.org/10.1023/A:1008935617754.

[30] Bo Thiesson, David Maxwell Chickering, David Heckerman, and Christopher Meek. Arma time-series modeling with graphical models. In David Maxwell Chickering and Joseph Y. Halpern, editors, UAI, pages 552–560. AUAI Press, 2004. ISBN 0-9749039-0-6. URL http://dblp.uni-trier.de/db/conf/uai/uai2004.html#ThiessonCHM04.

[31] David Poole and Alan K. Mackworth. Artificial Intelligence - Foundations of Computational Agents. Cambridge University Press, 2010. ISBN 978-0-521-51900-7.

[32] Robert Sedgewick and Kevin Wayne. Introduction to programming in Java - an interdisciplinary approach. Pearson / Addison Wesley, 2008. ISBN 978-0-321-49805-2.

[33] Pieter-Jan Van de Maele. Reading a imu without kalman: The complementary filter, April 2013. http://www.pieter-jan.com/node/11.

[34] Shane Colton. The balance filter, 2007. http://web.mit.edu/scolton/www/filter.pdf.

[35] Microsoft. Motion class, November 2013. http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.devices.sensors.motion(v=vs.105).aspx.

[36] Craig Lloyd. Microsoft has sold 4.2 million windows phone 8 handsets, according to facebook data, 2012. http://www.slashgear.com/microsoft-has-sold-4-2-million-windows-phone-8-handsets-according-to-facebook-data-10260224/.

[37] Steven M. LaValle. Planning algorithms, 2008. http://planning.cs.uiuc.edu/node102.html.

A Probability updating

Normal distribution for SF and CMA are hacks, and are not listed here. For SF, the variance is postulated to the sum of its parents' variance. For CMA, the variance is corresponding to the moving average it is copying.

Formula for the acceleration.

$$P(acc_{i+1}) = \mathcal{N}(\begin{bmatrix} accX_{i+1} \\ accY_{i+1} \\ accZ_{i+1} \end{bmatrix} \sigma^2_{acc}) \tag{A.1}$$

Formula for the gyroscope.

$$P(gyro_{i+1}) = \mathcal{N}(\begin{bmatrix} gyroX_{i+1} \\ gyroY_{i+1} \\ gyroZ_{i+1} \end{bmatrix}, \sigma^2_{gyro}) \tag{A.2}$$

Formula for aTan2.

$$\begin{aligned} P(aTan2_{i+1}) &= \mathcal{N}(G, \sigma^2_{acc}) \\ G &= \begin{bmatrix} atan2(\mu_{accY_{i+1}}, -\mu_{accZ_{i+1}}) \\ 0 \\ atan2(\mu_{accY_{i+1}}, -\mu_{accX_{i+1}}) \end{bmatrix} \end{aligned} \tag{A.3}$$

Formula for Comp.

$$\begin{aligned} P(Comp_{i+1}) &= \mathcal{N}(F * A, C) \\ F &= \begin{bmatrix} \alpha & \alpha * \Delta t & 1 - \alpha \end{bmatrix} \\ A &= \begin{bmatrix} \mu_{compX_i} & \mu_{compY_i} & \mu_{compZ_i} \\ \mu_{gyroX_{i+1}} & \mu_{gyroY_{i+1}} & \mu_{gyroZ_i} \\ \mu_{atan2X_{i+1}} & \mu_{atan2Y_{i+1}} & \mu_{atan2Z_i} \end{bmatrix} \\ C &= \sigma^2_{gyro_{i+1}} + \sigma^2_{atan2_{i+1}} \end{aligned} \tag{A.4}$$

Formula for the moving average of the acceleration.

$$\begin{aligned} P(MA_{i+1}) &= \mathcal{N}(F_{MA} * A_{MA_{i+1}}, G_{MA_{i+1}} + F_{MA} * C_{MA_{i+1}} * F^{\intercal}_{MA}) \\ &\text{where} \\ F_{MA} &= \begin{bmatrix} \alpha & (1 - \alpha) \end{bmatrix} \\ A_{MA_{i+1}} &= \begin{bmatrix} \mu(SF_{i+1}) \\ \mu(MA_i) \end{bmatrix} \\ C_{MA_{i+1}} &= \begin{bmatrix} var(SF_{i+1}) & 0 \\ 0 & var(MA_i) \end{bmatrix} \end{aligned} \tag{A.5}$$

Formula for the velocity.

$$\begin{aligned}
P(V_{i+1}) &= \mathcal{N}(F_V * A_{V_{i+1}}, G_{V_{i+1}} + F_V * C_{V_{i+1}} * F_V^\intercal) \\
&\text{where} \\
F_V &= \begin{bmatrix} \Delta t & 1 \end{bmatrix} \\
A_{V_{i+1}} &= \begin{bmatrix} \mu(MA_{i+1}) \\ \mu(V_i) \end{bmatrix} \\
C_{V_{i+1}} &= \begin{bmatrix} var(MA_{i+1}) & 0 \\ 0 & var(V_i) \end{bmatrix}
\end{aligned} \tag{A.6}$$

Formula for the position.

$$\begin{aligned}
P(Pos_{i+1}) &= \mathcal{N}(F_{Pos} * A_{Pos_{i+1}}, G_{Pos_{i+1}} + F_{Pos} * C_{Pos_{i+1}} * F_{Pos}^\intercal) \\
&\text{where} \\
F_{Pos} &= \begin{bmatrix} \Delta t & 1 \end{bmatrix} \\
A_{Pos_{i+1}} &= \begin{bmatrix} \mu(V_{i+1}) \\ \mu(Pos_i) \end{bmatrix} \\
C_{Pos_{i+1}} &= \begin{bmatrix} var(V_{i+1}) & 0 \\ 0 & var(Pos_i) \end{bmatrix}
\end{aligned} \tag{A.7}$$

B Precision Test

#	calculated position [cm]	actual position [cm]	%
1	42.22	100	42.22
2	47.27	100	47.27
3	51.77	100	51.77
4	44.58	100	44.58
5	40.69	100	40.69
6	41.61	100	41.61
7	53.56	100	53.56
8	51.86	100	51.86
9	114.24	100	114.24
10	34.98	100	34.98
11	35.35	100	35.35
12	38.93	100	38.93
13	43.92	100	43.92
14	37.65	100	37.65
15	42.73	100	42.73
16	38.21	100	38.21
17	40.81	100	40.81
18	48.23	100	48.23
19	29.52	100	29.52
20	41.94	100	41.94
21	33.87	100	33.87
22	35.15	100	35.15
23	48.72	100	48.72
24	23.28	100	23.28
25	35.69	100	35.69
26	163.06	100	164.06
27	97.48	100	97.48
28	27.40	100	24.40
29	38.96	100	38.96
30	36.77	100	36.77

#	calculated position [cm]	actual position [cm]	%
31	-31.24	-100	31.24
32	-47.94	-100	47.94
33	-44.66	-100	44.66
34	-23.72	-100	23.72
35	-35.48	-100	35.48
36	-28.24	-100	28.24
37	-44.79	-100	44.79
38	-33.05	-100	33.05
39	-140.18	-100	140.18
40	-30.05	-100	30.05
41	-35.48	-100	35.48
42	-42.26	-100	42.26
43	-43.27	-100	43.27
44	-39.82	-100	39.82
45	-35.68	-100	35.68
46	-43.84	-100	43.84
47	-46.01	-100	46.01
48	-56.75	-100	56.75
49	-57.65	-100	57.65
50	-41.21	-100	41.21
51	-58.86	-100	58.86
52	-41.77	-100	41.77
53	-45.26	-100	45.26
54	-41.33	-100	41.33
55	-35.27	-100	35.27
56	-28.92	-100	28.92
57	-35.78	-100	35.78
58	-22.79	-100	22.79
59	-29.27	-100	29.27
60	-36.06	-100	36.06
61	-33.99	-100	33.99
62	-46.24	-100	46.24

#	calculated position [cm]	actual position [cm]	%
63	98.83	200	49.42
64	95.29	200	47.65
65	79.20	200	39.60
66	96.35	200	48.18
67	82.30	200	41.15
68	105.42	200	52.71
69	-90.29	-200	45.15
70	-76.60	-200	38.30
71	-93.50	-200	46.75
72	-59.20	-200	29.60
73	-73.03	-200	36.52
74	-173.48	-200	86.74
75	1020	2500	40.80
76	1031	2500	41.24
77	1360	2500	54.40
78	1177	2500	47.08
79	-1551	-2500	62.04
80	-1582	-2500	63.28
81	-784	-2500	31.36
82	-909	-2500	36.36
83	2043	5000	40.68
84	2336	5000	46.72
85	3066	5000	61.32
86	-3557	-5000	71.14
87	-1695	-5000	33.90