Position Tracking using a self learning mobility model

Group 11gr1001

Troels Laursen
Nikolaj Pedersen

Aalborg University, Spring 2011

AALBORG UNIVERSITET

The Faculties of Engineering, Science and Medicine
Department of Electronic Systems
Frederik Bajers Vej 7
Phone: +45 96 35 86 00
http://es.aau.dk

Title: Position Tracking using a Self Learning Mobility Model
Theme:
Design of distributes systems
Project period:
February 1st - June 1st, 2011

Project group: 11gr1001

Group members:
Troels Laursen
Nikolaj Pedersen

Supervisors:
Assoc. Prof. Tatiana Kozlova Madsen
Stud. Ph.D. Jimmy Jessen Nielsen

Number of copies: 5
Number of pages: 43
Appended documents:
(X appendix, 1 CD-ROM)
Total number of pages: 66
Finished: June 2011

Abstract:

The objective of this Project is to test different approaches for filtering position measurements in a positioning system when tracking a moving object. Several different filters was tested based on mobility models of different complexity.
The first filter used does not utilizi any mobility model, the second filter uses movement speed, the third filter uses movement speed and direction in an attempt to favor straight line movement. The last filter we propose is an adaptive version that uses training measurements to learn movement patterns and thereby pay regards to any geographical depended mobility changed.

All the filters is tested in a variety of different simulations where a simple noise model is used. The adaptive model is found to be best, an improvement of 15% in off-line mode and 19% in on-line mode when comparing with the directional mobility model.

The adaptive model was tested with different variance of noise which give some results that needs to be investigated further.

As a further development some runtime optimizations is interesting and should be possible.

Chapter 1

Preface

This report is the documentation of the project concerning position determination in noisy environments using training data. The project is conducted by two 10^{th} semester students of Networks and Distributed Systems at Department of Electronic Systems.

Determination of position is something almost every cellphone can do today. However, in most cases GPS is used. The problem with GPS is the use of satellites which means it does not work indoor, and has trouble working in the centre of big citeis due to the many high-rise buildings.

1.1 Study Programme

The project is in line with the curriculum which says 'skills thought in previous courses must be used'. This project involves theory from Graph theory, Markov chains and probability theory.

1.2 Notes and References

The report makes use of cross references. Figures, tables, and source code excerpts are numbered, and these numbers are used in the text. In some cases, the size of the figures and tables makes it necessary to place it on the following page. When abbreviations is used first time the full name is written with the abbreviation in brackets thereafter only abbreviation is written, a list a abbreviations can be found in appendix D on page 63.

When using external sources, information about the source is given in the bibliography in appendix D on page 65. The source information is referenced by a text and year encapsulated in brackets. A reference to the example source is [Example 08b].

1.2.1 Author Signatures

Nikolaj Bisgaard Pedersen

Troels Laursen

Table of Contents

Chapter 2

Introduction

Tracking various items and even people has for years been the objective for many engineers and scientists. The result of this great interest has been a variety of different tracking technologies e.g. LOng RAnge Navigation (LORAN), Global Positioning System (GPS), Intel Precision Location Technology (PLT) and Cambridge Positioning System (CPS) [Hoo11].

Traditionally positioning systems were reserved for military applications and safety related ship and aircraft borne systems. Ship borne safety related systems used LORAN for determining location and speed, but since *selective availability* for GPS was turned off and high quality signals became available for civilian use these systems shifted to the use of GPS for positioning. Many systems are now using GPS for positioning, but also many new Mobile Device (MD) are equipped with GPS [Wik11].

GPS is in a number of situations not working well enough for the standard MD applications due to obstructed Line Of Sight (LOS) with the satellites for example indoors and in dense urban areas where tall buildings can obstruct LOS.

2.1 Indoor Position Systems

Indoor positioning systems are still used in more and more applications. Many museums, shopping malls and airports are currently investigating the possibility of using indoor positioning systems for guidance, geographical advertisements and costumer behavior statistics.

The mentioned limitations of GPS of indoor use has caused many different approaches for using other system than GPS when indoors. Some of the approaches is based on different radio technologies e.g. Bluetooth, Wireless Fidelity (Wi-Fi) and Radio Frequency IDentification (RFID). The advantage of using these short range technologies is that in many urban areas and buildings they are already present solving other tasks. The use of short range technologies for assisting GPS in indoor and urban areas can be achieved by many different methods e.g. Radio Signal Strength (RSS), Time Of Arrival (TOA), Angle Of Arrival (AOA) and Time Difference of Arrival (TDoA) [SCGL05], [STK05], [GG05], [PAK+05]. Some of them can be used without disturbing the normal use of the short range radio technology. Common for them all is the requirement of multiple access points within reach of the MD.

All of these technologies will deliver a position that is subject to various amounts of noise on the measurement. In a Bluetooth setup using RSS an average error of 90 cm was measured[RPN+08].

An example of a combined guidance and geographical advertisement system is the SITA developed indoor Wi-Fi based positioning system implemented in Copenhagen International Airport. The system has an accuracy of approximately three meters and besides helping passengers find their way around the airport, and give them geographical information it also helps airport officials improve the design of the airport, direct the flow of passengers and shift employees to improve security matters [Neg11].

Another example of a working indoor positioning system is the MediaCart. The MediaCart is a shopping cart that by using a RFID scanner scans the proximity for RFID tags and thereby becomes aware of it is location, which then is used to give the shopper adverts about products in the proximity of the shopper. The shopper can make an online shopping list which then is displayed on a screen on the shopping cart together with a map showing where the next item is located compared to the shoppers location. The system is also used to give the advertisers statistical information of their products and the effectiveness of their adverts [Med10].

2.1.1 System Overview

A short range radio technology positioning system can be divided in four independent parts. One part concerns what type of radio technology (e.g. Bluetooth, Wi-Fi and RFID) and how these technologies are used (e.g. TDoA, TOA and RSS). The next part concerns converting the incoming measurement to actual positions which can be done in a number of ways depending on how these measurements are obtained (e.g. Sensor fusion and fingerprinting). The third part is the filtering or smoothing part whose purpose is to attenuate the noise. The last part is the application, which can be everything from an indoor guidance system to supervision of people or animals. This is illustrated in Figure 2.1.

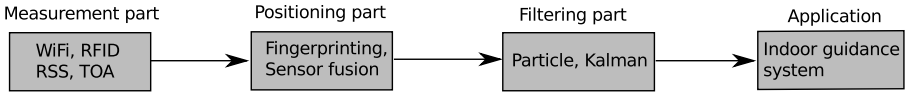

Figure 2.1: Block diagram of a short range radio technology positioning system. From radio signals to a indoor guidance system. Each box includes some examples of what type of technology they can consist of

The part about filtering and smoothing will be the focus of this report.

Initial Problem Definition

How can filtering or smoothing of the measurements improve the accuracy of the position.

2.1.2 Mobility in systems

Indoor positioning systems based on short range radio technologies will give noisy input measurements which is why filtering or smoothing is required. The filtering or smoothing can be done in various ways, but the users behaviour can often be used to increase the performance for the filter. If for instance the object is a train the movement would be very straight with few soft turns and speed changes. However, if the user is a pedestrian the probability of changing course and speed will be higher. This information of the user is known as a mobility model and can generally be divided in three categories[NHM+11].

- State-space models
- Memoryless random models
- Geographical models

State-space Models

A state-space model is a model which utilizes kinematic parameters such as position, linear and angular velocity, accelerations, etc. These parameters are used to form a state vector that describes the physical possible trajectory of the user. This state vector can be used to restrict the types of allowable maneuvers. If for instance the input measurements suggest that a pedestrian has moved 100 meters in one second, which is probably not the case but a result of measurement noise.

An example of such a model is the constant velocity model where the allowable velocity for the object is restricted by a maximum and a minimum velocity. Other examples are the improved version of the constant velocity model, the less drunk model[RPN+08] where the trajectory angle is restricted to form more straight paths.

Memoryless Random Models

Memoryless random models are simple and allow rapid change in velocity and direction. They do not restrict the dynamics to what is possible as state-space models do. An example of a memoryless random model is the random waypoints model where every possible outcome of all dynamics are allowed, which means that the object can travel with unlimited speed. Another model is the Brownian motion, which describes how a particle moves in fluids.

Geographic Models

Geographic models are unlike state-space models not defined for the object but for some geographical points. Geographic models can for instance decrease the probability for the object passing trough obstacles and adjust the speed for some areas. An example of a geographic model is the pathway mobility model which increase the probability for the object to travel within some specified paths. The geographic model is sometimes used for map matching.

These mobility models are then usually used by a particle filter or a Kalman filter sequentially with the incoming position measurements[RPN+08], [Thr00] and [GB00].

2.2 Motivation

The most common model to use when filtering position measurements are a state-space model based on expected behavior of the object. This is the obvious choice when the object is known. However, in for example indoor environments where walls and furniture delimit user movements, it is beneficial with a geographical mobility model that knows which movements are possible and typical at certain geographical points. However, obtaining this information of the most probable behavior can be a tedious task of typing probable behavior for each geographical point in a map. This project suggests a self learning geographical mobility model that dynamically sets the typical behavior of the object for all points in a map. This means for instance if the user typically changes speed or direction on a certain geographical point the self learning algorithm will eventually learn this behavior and include it in the mobility model for the geographical point.

2.2.1 Refined Problem Definition

How can filtering or smoothing of the measurements improve the accuracy of the position by using a mobility model describing the user behavior and through the use of an self-learning geographical dependent mobility model.

Chapter 3

Proposal

Noise is in many positioning systems a hurdle that has to be addressed [RPN+08], [SCGL05] and [GG05]. This chapter will address the issue of noisy position measurements. For simplicity reasons the possible positions is limited to fall within a certain grid of cells (occupancy grid map) [Mil97]. This means that a position measurement is moved to be at the nearest grid cell. This goes for both the noisy data and the actual positions. Furthermore it has been chosen yet again for simplicity reasons that the only possible moves for the object which is tracked is to a neighboring grid cells.

Many methods for removing or reducing noisy position measurements have been proposed [RPN+08] and [GG05]. This chapter starts off by proposing the use of a Moving Average filter to do a simple smoothing of the noise (Section 3.1). Then by using a Hidden Markov Model (HMM) to model the behavior of the tracked object (see Section 3.2). We propose using Viterbi to find the most probable path (see Section 3.3). The Viterbi algorithm is then reduced in complexity (see Section 3.4). The model used for the Viterbi and the reduced Viterbi algorithms are assuming equal probabilities for the object to move in all directions. This is changed in the directional Viterbi to be depended from the state before so that the probability for going in a straight line is increased (see Section 3.5). Finally, we propose a Viterbi algorithm that uses an adaptive model which changes according to movement patterns in some training trajectories (see Section 3.6)

3.1 Moving Average (Filter One)

One of the simplest filters that can be used to filter or smooth is the Moving Average. In Eq. (3.1) the Moving Average in general terms is presented where Z is the noisy input and X is filtered or smoothed. For this project a Moving Average with filter order of 4 is chosen. The filter order of four is believed to be the best compromise in the trade-off with delay and smoothing performance.

$$
\begin{aligned}
X(n) &= \frac{Z(n) + \cdots + Z(n-i+1)}{i} && && (3.1)\\
X(1) &= Z(1) && i = 4\\
X(2) &= \frac{Z(2) + Z(1)}{2} && i = 4\\
X(3) &= \frac{Z(3) + Z(2) + Z(1)}{3} && i = 4\\
X(4) &= \frac{Z(4) + Z(3) + X(2) + Z(1)}{4} && i = 4\\
X(5) &= \frac{Z(5) + Z(4) + X(3) + Z(2)}{4} && i = 4\\
&\vdots
\end{aligned}
$$

The way the first five values are calculated is shown after Eq. (3.1). This is done for both X-coordinates and Y-coordinates separately and fitted to the closest grid-point. A Moving Average filter works as a low-pass filter, which will remove some of the fluctuation of the noisy inputs, but also add a delay which eventually leads to errors.

The Moving Average filter does not use any knowledge about how the tracked object moves. Knowledge about the tracked object can improve the filter performance [RPN+08], [SCGL05] and [PAK+05].

3.2 Mobility Models

A model of the system creating the measurements is needed in order of implementing a filter that by using this extra information increases the accuracy of the positions. This model will in the case of a positioning system consist of both a mobility model and a noise model that hides the actual positions. Because of the choice of using a grid, it is logical to use a HMM [RPN+08] to model the behaviour of an object. A HMM consist of four parts: a set that contains all the different states in the Markov model, a matrix containing the different probability to change from one state to another, an emission part that contains the set of possible measurements and a matrix that contains the probabilities of a particular measurement depending on the state.

The state part, which is all the different grids, is also the hidden part which is desired to estimate. All the states are given names as in a X-Y-coordinate system. The set containing all the states is called Z. Because every state has two variables, a transition probability needs to have four index-variables, for instance from state 6,7 to state 6,8 $a_{6,7,6,8} = 0.0625$. This is then arranged in a transition probability matrix called A since four dimensional matrices are hard to draw two of the variables are locked in the matrix in Eq. (3.2) (here 6,7 is the locked values). Most of the values in Eq. (3.2) are zero because in the chosen model, it is only possible to move to neighboring grid cell. The transition probability matrix, A, is what is needed to make a Markov model. Making it hidden requires the emission set and the emission probabilities.

$$a_{6,7,:,:} = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \cdots & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots \\ \cdots & 0.0 & 0.0625 & 0.0625 & 0.0625 & 0.0 & \cdots \\ \cdots & 0.0 & 0.0625 & 0.5 & 0.0625 & 0.0 & \cdots \\ \cdots & 0.0 & 0.0625 & 0.0625 & 0.0625 & 0.0 & \cdots \\ \cdots & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.2}$$

The measurements are also positions consisting of a X and Y component since these are dependent on the actual position(state). The emission probabilities are also arranged in a four dimensional matrix called B. They are written slightly different because the emission probabilities are more like a function of the measurement (observation) $b_{state}(obs)$. The B matrix depends on the noise model and differs depending on the noise. In some cases the probabilities of beginning in the different states π is provided, but in other cases they will have to be calculated from the transition matrix A. When modeling positions the probabilities of the beginning state is an uniform distribution.

To find the best set of state transitions and output probabilities some kind of algorithm is needed.

3.3 Viterbi (Filter Two)

On of the ways to find the most probable path of states in a HMM when only a set of measurements is available is to use the Viterbi algorithm [JM00]. The Viterbi algorithm basically works around two equations, Eq. (3.3) and Eq. (3.4). The first equation, Eq. (3.3), used to initialize the Viterbi algorithm. The state the calculations are done for is denoted k. x_0 is the first emission and π_k is the probability of beginning in state k. V is the probability of being in that state at time t.
The main part of the algorithm is the second equation, Eq. (3.4), Z is a set containing all the states, and t is the time variable as moving through the observations.

$$V_{0,k} = b_k(x_0) \cdot \pi_k \tag{3.3}$$
$$V_{t,k} = b_k(x_t) \cdot MAX_{z \in Z}(a_{z,k} \cdot V_{t-1,z}) \tag{3.4}$$

This then of course has to be done of all k in states.
At the end of and observation sequence the most probable state is found using Eq. (3.5).

$$z_T = MAX_{z \in Z}(V_{T,z}) \tag{3.5}$$

Where T is the number of measurements.

This needs to be back traced in order to give a path instead of just the most probable end state. To do this the state z in Eq. (3.4) that yielded the highest value is saved.

Algorithm and Complexity

When written like an algorithm Eq. (3.3) becomes Algorithm 1. This algorithm's complexity is only dependent on Z the number of states $O(Z)$.

```
for z in Z do
    trellis[0][z] =π_z · b_z(obs[0])
    path[0][z] = z
end
```

Algorithm 1: Eq. (3.3) written as an algorithm

Because Eq. (3.4) needs to be sweeped through the entire measurement, an extra loop running through t is added.

```
for t in range(1,length(obs)-2) do
    for z in Z do
        for x in Z do
            (newprob = trellis[t-1][x]·a_xz · b_z(obs[t]))
            if newprob > prob then
                prob = newprob
                state = x
            end
        end
        trellis[t][z] =prob
        path[t][z] = state
    end
end
```

Algorithm 2: Eq. (3.4) written as an algorithm

To calculate the max probability in Eq. (3.4) a sweep through all the possible values is done; thus complexity $O(Z)$. Because this is inside the other two loops, the total complexity is $O(T \cdot Z^2)$. The last part of finding the most probable walk is done with Algorithm 3 that sim-

ply back traces from the most probable end state. This functionality gives a complexity of $O(Z+T)$.

```
state = max(trellis[length(obs)-1][z],z) for z in Z
walk[length(obs)] = state
for t in range(1,length(obs)-2) do
    walk[length(obs)-t]=path[length(obs)-t+1][state]
    state= walk[length(obs)-t]
end
return walk
```

Algorithm 3: The terminate phase of the Viterbi algorithm

The complexity of the total algorithm is $O(T \cdot Z^2)$ which comes from Eq. (3.4). The variables are the amount of states Z and the length of the observation sequence T. This makes the algorithm very computationally heavy because the number of states are high, which makes the algorithm very heavy for the task of finding the way through a large map. In Appendix B the calculations of running a simple example through the Viterbi algorithm.

3.4 Reduced Complexity Viterbi (Filter Three)

It has been realized that the normal Viterbi algorithm was very computationally heavy when used to find a path through a map. It has also been realized that many of the calculations the algorithm has been performing where multiplication operations where at least one of the values where zero or very close to zero. In other words, this was an obvious place for improvement.
The first part was to look at all the zeros in the mobility model and instead add a functionality that finds all the non zero values; this function is called neighbors. The complexity of the neighbors function has to be considered and needs to be designed differently depending on the mobility model. The use of neighbors reduces the complexity to $O(T \cdot Z \cdot N)$ where N is the number of states it is possible to arrive from. This means the transition matrix A can be reduced in size so that the probability for a transition from one particular state to another state can be a 3x3 matrix as shown in the matrix in Eq. (3.6). This optimization is at the cost of some extra calculations, which is a trade-off between whether memory usage or runtime. This optimization is not done in Algorithm 4 but is done in Algorithm 6.

$$a_{6,7,:,:} = \begin{bmatrix} 0.0625 & 0.0625 & 0.0625 \\ 0.0625 & 0.5 & 0.0625 \\ 0.0625 & 0.0625 & 0.0625 \end{bmatrix} \tag{3.6}$$

The second possible optimization is to make the first loop dependent on the noise. It is not necessary to calculate the possibility for all states only the ones close to the measurement i.e. the states where the probability in the B matrix is above 10^{-14}. Using a variance of one gives a 15x15 matrix. Only the trellis values for this 15x15 matrix is calculated instead of the entire map. These are all changes taking places in Algorithm 2 and is shown in Algorithm 4.

```
for t in range(1,length(obs)-1) do
    N= noise_states(obs[t])
    for z in N do
        S = neighbors(z)
        for x in S do
            (newprob = trellis[t-1][x]·a_xz · b_z (obs[t]))
            if newprob > prob then
                prob = newprob
                state = x
            end
        end
        trellis[t][z] = prob
        path[t][z] = state
    end
end
```

Algorithm 4: The reduced complexity Viterbi Algorithm

If these two reductions in complexity is applied in cases significantly different than this it might increase runtime and introduce errors. Special considerations has to be placed at the functions noise_states(obs[t]) and neighbors(z) as they need to have a low complexity due to the high number of calls.

3.5 Directional Viterbi (Filter Four)

The filtering method proposed in [RPN+08] is a particle filter with a mobility model based on a HMM. The model is describing a pedestrian where the probability for a straight trajectory is increased and the probability for a fluctuating trajectory is decreased.

In an attempt to improve the performance of the Viterbi algorithm, a modification is made to account for the direction the tracked object is heading. This is done by looking at where the object came from and update the A matrix so the probability for going further in the same direction is higher. This means that the A matrix gains two extra dimensions.

The movement in Figure 3.1 is a $-1, +1$ movement.

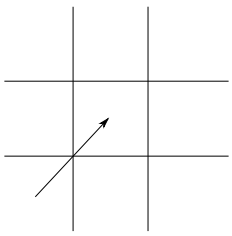

Figure 3.1: Example of a $(-1, +1)$ movement

A 3x3 matrix exists for all 9 possible movements. The A matrix for the case illustrated in Figure 3.1 is shown in Eq. (3.7). The used A matrix is shifted after each step if for instance the step is to the current cell (meaning "not going anywhere") the A matrix from last step is used. The first step which is the movement case $(0, 0)$ has an A matrix identical to the standard A matrix from Section 3.4 Eq. (3.6).

$$a_{6,7,0,2,:,:} = \begin{bmatrix} 0.05 & 0.125 & 0.15 \\ 0.0 & 0.5 & 0.125 \\ 0.0 & 0.0 & 0.05 \end{bmatrix} \tag{3.7}$$

As seen in Eq. (3.7) the possibility of the next state being backwards is illuminated. The trajectories are not expected to be completely straight; thus the probability of making a slight turn is set to be almost equal to going straight.

Including knowledge about the previous state increases the memory usage, which is why the previous mentioned memory optimization from Section 3.4 is applied here as well.

The directional version of the Viterbi algorithm differs from the Viterbi algorithm. Both the initial and the main part of the algorithm is changed. The new algorithms can be seen in Algorithm 5 and in Algorithm 6

```
for y in Y do
    trellis[1][y] =π_y · b_y(obs[0])
    path[1][y] = y
    dir[1][y] = (2,2)
end
```

Algorithm 5: Modified version of Algorithm 1 to also include direction

```
for t in range(2,length(obs)) do
    for y in Y do
        S = neighbors(y)
        for x in S do
            dif =y+1-x
            (newprob = trellis[t-1][x]·a_{x,dif,dir[t−1][x]} · b_y(obs[t]))
            if newprob > prob then
                prob = newprob
                state = x
            end
        end
        trellis[t][y] =prob
        path[t][y] = state
        if path[t][y]=path[t-1][y] then
            dir[t][y]=dir[t-1][y]
        else
            dir[t][y] = y+1-state
        end
    end
end
```

Algorithm 6: The Directional Viterbi Algorithm

The directional version of Viterbi should improve the performance when tracking objects with a straight trajectory with minor soft turns, but the algorithm is expected however have some problems with sharp and sudden turns.

3.6 Adaptive Viterbi (Filter Five)

An attempt to improve the algorithms ability to remove the noise is done by making a version where the transition matrix A is adaptive. It takes basis in the directional Viterbi in Section 3.5 but with some added extra functionalities. These extra functionalities update the transition matrix A for a given state (grid point) so it fits the statistical behaviour for the state (grid point). This means for instance if some amount of training trajectories shows a significantly high probability for changing course on a specific state (grid point). The A matrix is updated accordingly.

The training data for the filter will have to be contiguous in the sense that each step between grids has be to a neighboring grid point. This requires some filtering or a composition algorithm for the training data.

Two versions of the adaptive Viterbi algorithm is proposed: one that is self learning adaptive where the output from the adaptive Viterbi algorithm is used to update the transition matrix A. This functionality is shown in Figure 3.2. The self learning adaptive version uses itself to filter the training trajectories so they are contiguous. The self learning algorithm updates sequentially with

the incoming positions. This means in practice that the self learning algorithm can be implemented and update itself sequentially with filtering.

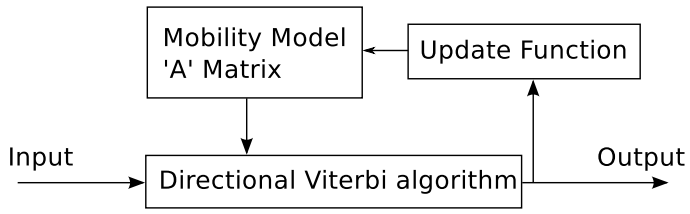

Figure 3.2: The adaptive Viterbi algorithm self updating version

The other version of the adaptive Viterbi uses some external function to connect the training data in a trajectory and then uses the update function to update the A matrix off-line, meaning that this, the external, version of the adaptive Viterbi algorithm updates the A matrix before implementation. The external adaptive Viterbi is shown on Figure 3.3. When the external adaptive Viterbi is filtering no feedback is fed to the algorithm, which means that no matter how the filtered data acts the A matrix is not updated. However, this can be seen as an advantage due to the fact that the self learning adaptive Viterbi potentially can be unstable.

The composition part can either be a filter that makes a trajectory and suppresses noise, or it can be some algorithm that connects the noise in a trajectory. An investigation on whether to use a filter or just a connector will have to be done.

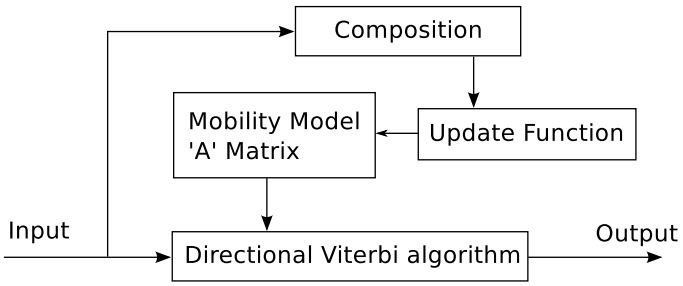

Figure 3.3: The adaptive Viterbi algorithm external updating version

The Directional Viterbi algorithm is identical to the previous one described in Section 3.5. The new thing here is the update function which can be implemented by using a forgetting factor ff. The update function takes contiguous trajectory and uses it to update the transition matrix A. In order to update a transition matrix element like Eq. (3.8) the previous position is needed to find the right element, and the next position is needed to know which value to increase.

Transition matrix element before update:

$$a_{6,7,0,2,:,:} = \begin{bmatrix} 0.05 & 0.125 & 0.15 \\ 0.0 & 0.5 & 0.125 \\ 0.0 & 0.0 & 0.05 \end{bmatrix} \tag{3.8}$$

Transition matrix element after update with a forgetting factor of 0.05:

$$a_{6,7,0,2,:,:} = \begin{bmatrix} 0.0476 & 0.119 & 0.191 \\ 0.0 & 0.476 & 0.119 \\ 0.0 & 0.0 & 0.0476 \end{bmatrix} \tag{3.9}$$

This can the be written as an algorithm. See Algorithm 7.

for *t in range(2,length(path)-1)* **do**
  x = path[t]
  y = path[t]-path[t-1]+2
  z = path[t+1]-path[t]+2
  $a_{xy}(z) = a_{xy}(z) + ff$
  $a_{xy} = \frac{a_{xy}}{1+ff}$
end

Algorithm 7: The Update Function

Some consideration has to go into the ff because if it is too high, the filter will not converge but if it is too low it will take a very long time for it to converge.

Chapter 4

Evaluation

In order to test the performance of the filters described in Chapter 3 either measurements or simulations have to be preformed. A real life measurement would require a full test setup with Access points, a user and a large room with space for all the different tests. Furthermore the test setup would require distance measurements of the user in order of obtaining actual paths will be needed in order to evaluate the performance of the filters. The adaptive filter also requires a large number of training paths for it to do the self learning. To do both these tasks will be very time consuming.

The noise characteristics in a real life measurement will depend on a number of factors like: walls, obstacles, humidity, temperature etc. and a variety of different factors depending on which method the positions are obtained with (e.g. RSS, TOA and TDoA). The advantage of a real life measurement is at the same time the drawback for the measurements. Results from a real life measurement would be valid and a good test for the filters, but it is only a test for the specific noise characteristics of that setup.

Simulations, however, is a good opportunity for a controlled test with the opportunity of changing a lot the parameters and test the filters with different noise characteristics. The actual path and training paths are very easy to obtain.

For this project the chosen approach for testing the filters is simulation. This has been approach chosen because of the opportunity for testing the filters with different noise characteristics and due to fact that real life measurements are very time consuming.

4.1 Simulation Setup

When measuring performance for the filters various test scenarios are used. It is chosen that the filters should be applicable for a pedestrian in an office environment.

Four different scenarios are chosen to evaluate the filter performance for the individual filters. The four different scenarios are showed and explained below.

- Straight line
- 90° turn
- 180° turn on a straight line
- Soft turn(Circle)

The four scenarios are illustrated in Figure 4.1

The four different scenarios found in Figure 4.1 are chosen because they are considered the components in a pedestrian's walking pattern. The filters are tested with these four different scenarios in order to investigate the differences in the filters when filtering these paths.

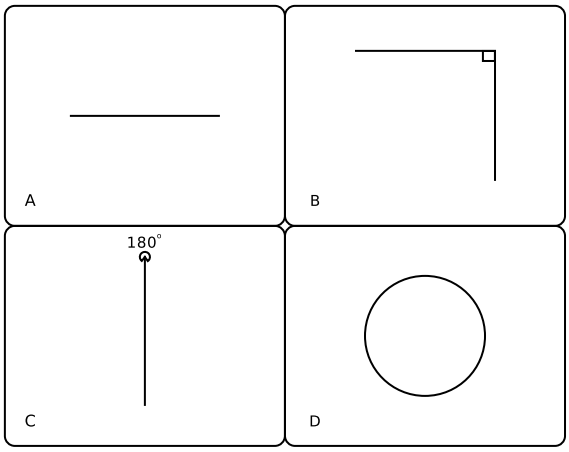

Figure 4.1: The four different scenarios. A is the straight line, B is the 90° turn, C is the 180° turn and D is the soft turn(circle)

4.2 Pedestrian Simulation

The four test scenarios are tested individually as components in how the movement pattern for a pedestrian looks like, but a test of a pedestrian is also needed.

The simulations of a pedestrian are done by selecting a map and divide the map in a grid of $50X50$ squared cells illustrated in Figure 4.2. Each cell represent a possible position which in this case gives 2500 different positions (except from the walls). The green lines together with the border of the illustration are walls.

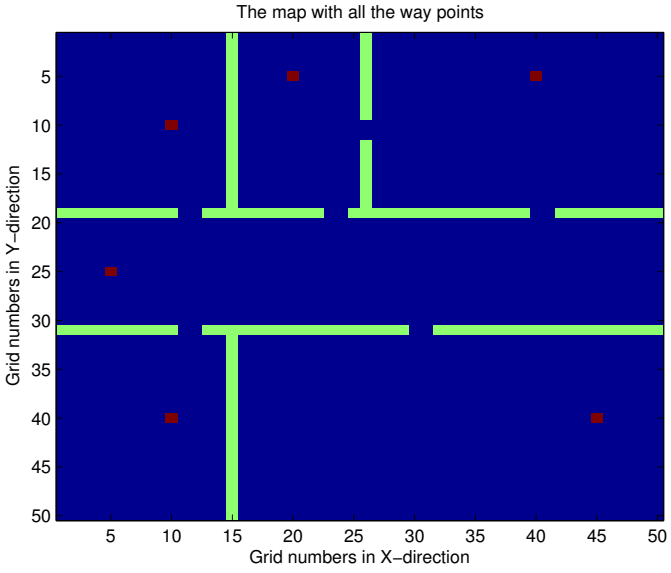

Figure 4.2: The map used to simulate the movement of a pedestrian. The green lines are walls and the red squares are start positions.

To make the number of possible jumps from one cell low it is decided that the only possible move is to stay or move to a neighboring cell.

The red squares in Figure 4.2 are the chosen way points. A pedestrian walking from one point to another will walk the shortest path with few direction changes. The chosen trajectories for

the pedestrian is the shortest path between all of the red squares which gives 30 different paths. Furthermore, four paths running from one way point through another way point and to an end way point are added which gives a total of 34 paths.

The shortest path from all way point to all way points are found by using a path search algorithm.

4.2.1 Path Search Algorithm

The path search algorithm has to find the shortest path between two points and do it within a reasonable time. The Figure 4.3 shows how the map is converted to a graph which enables the use of path search algorithms in order to find a path between two or more points on the map. The blue dot on Figure 4.3 (a) is indicating the starting point where the grey neighboring nodes are the possible first move from the starting point. The Green dot is the next step where all the grey neighboring nodes are the possible second move. This is illustrated on Figure 4.3 (b) where the blue dot is represented as a blue square at the root. The second step is the green square. Both the green and the blue squares are connected with their neighboring nodes i.e. the possible moves from the green and the blue node respectively.

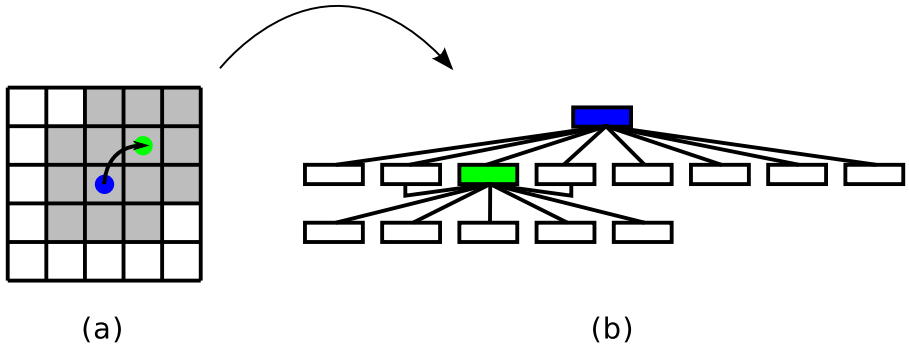

Figure 4.3: This figure illustrates how the map is converted to a graph. (a) shows a map where the blue dot is the root, and the green is the next root on the path. (b) shows how the graph looks like with the blue node as the first root and then the green as root. In (a) the grey squares are the neighboring squares to the blue root and the green root.

It is obvious that for a $50x50$ map the complete graph will become big and complicated so it is desirable to find an algorithm that can find the shortest path without having to explore the entire map.

Depth-First search

The depth-first algorithm starts at the root and explores the map by going as far as possible without visiting previous visited nodes. When the algorithm has reached a dead-end it starts to backtrack, which means going one step back and going another way; again without visiting nodes that have already been visited. Depth-first can be used to explore an entire map, but it takes time linear to the size of the map, and in case of a $50x50$ map it will take a lot of time before it will find the goal node. However, even after it has found the goal node there is no guarantee that it has found the shortest path. There is now way to guarantee the shortest path when using Depth-first[Joh].

Breadth-First search

The Breadth-first algorithm is much like the depth-first algorithm. The difference lies within the way it explores. Breadth-first starts at the root and finds all the children for the root and then

explores all the grand children and so on. Breadth-first has the same exploring capabilities as depth-first with regards to time. This means that in case of a $50x50$ map it will take a lot of time to find the goal node. The only way to be sure that we have found the shortest path is to explore the entire map, which is very time consuming[Joh].

A*

The A* search algorithm is unlike Depth-first not just exploring a random way and unlike Breadth-first exploring every possible route. The A* algorithm explores the most promising way first by a heuristic evaluation function. The evaluation function is a sum of the cost from the starting node to the current node and a distance measure from the current node to the goal node. Often the distances are calculated using the euclidean distance or Manhattan distance. A* guaranties the shortest path solution and because of the prioritisation done by the heuristic evaluation A* often finds the shortest path within a reasonable amount of steps, however, the worst case performance is not very good.[Les05]

4.2.2 Comparison

In order to compare the three algorithms the pro and cons of the different algorithms is investigated.

The Depth-first algorithm is very good in terms of best case search times. This is due to the fact that if Depth-first finds the goal in the first path (meaning no backtracking) it will go very fast also in the case of worse case it will only have to visit the every node once. Depth-first is very computationally easy and does not require a lot of calculations which effects the search time even in the case of some very long detours and backtracking. One of the biggest problems are that there is no guaranties for the shortest path or a very logical path. Actually, a dept-first is excellent for generating mazes if set to explore the entire map.

The Breadth-first algorithm will in have to explore the entire map if it is to guarantee the shortest path when operating with different edge values, this makes it very computational heavy for a big map. If operating without edge values, best case is when the goal is reached. Worst case is always exploring the entire map.

The A* algorithm is more complicated than the other algorithms because it requires a heuristic function. Nonetheless, the best case will have a high search time because the algorithm will still have to calculate the heuristic function for each step which will take some time. The worst case search time is very high because of the calculations done by the heuristic function and the fact that the worst case would be exploring the entire map. Unlike the other algorithms they will often end up close to the best case depending on the map if the map is a maze A* will preform very bad, but for a map with a low number of obstacles it will preform very well. One of the biggest advantages of this algorithm is its guarantee to find the shortest path and good scaling for big maps.

Based on the above mentioned comparison A* is chosen because it can guarantee the shortest path plus it will work well for a map containing obstacles.

The A* algorithm is described and implementation is explained in Appendix A. On Figure 4.4 a path found by the A* algorithm is illustrated.

The path found by A* does not have any fluctuation in the velocity. A* finds the shortest path which means that no cell is visited more then once. By introducing a function which runs through each step in the path and with a probability of 0.5 inserts a step to the same cell, the velocity is changed. This gives the probability for staying in the same cell of 0.5.

4.3 Noise Parameter

The input noise characteristics for a positioning system depends on various factors like the technology, the method and other factors like humidity and temperature.[GG05] If for instance the method is RSS the mean and variance of the noise would depend on where the actual position is due to walls, furniture and other things that can obstruct LOS or give reflections. With another method like TOA the noise mean and variance can vary due to timing issues like if the transmitter and the receiver are asynchronous in time and if the transmit time or receive time takes longer

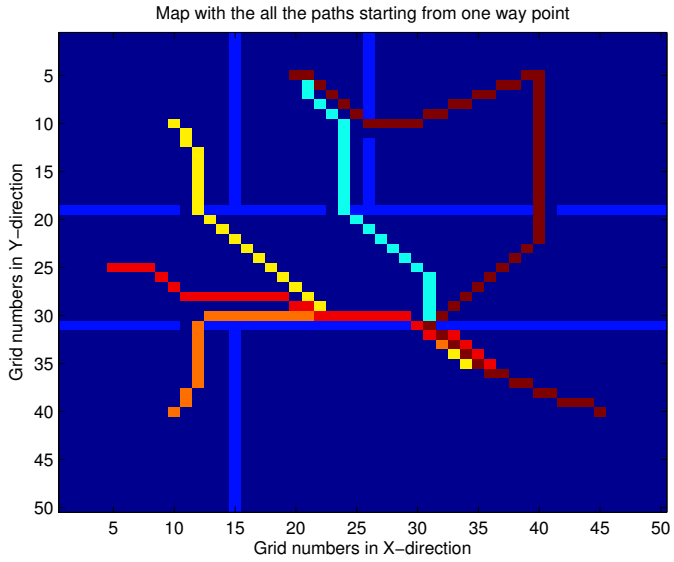

Figure 4.4: This figure shows the all the paths starting from one way point. Note that there are six different paths where one of them are hidden behind another (from the start to the upper right corner)

then anticipated. Not only the method can introduce deviation in the noise mean and variance, but also the technology can introduce some parameters for the noise like if the technology is used for other applications and other factors like the antenna and amplifier. If the noise model has to account for all these factors the model will become very complicated, and it would be very hard to compare results. Therefore it is chosen to model the noise as a normal distributed random variable which also is proposed in [PAK+05]. The noise is chosen to be additive because additive noise has the highest significance as stated in [PAK+05].

After noise has been added the positions from the positioning system (see Section 2.1.1, Figure 2.1) becomes observable for the filtering part. The observable sequence is illustrated in Figure 4.5 as $X(n)$

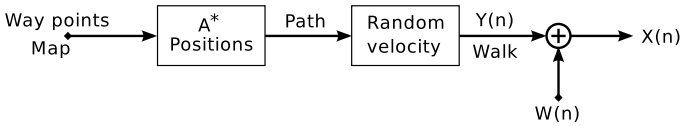

Figure 4.5: Block diagram for the simulation model. The observable sequence $X(n)$ and the non-observable sequence which is the actual positions $Y(n)$ (walk). The noise is denoted $W(n)$

The observable sequence $X(n)$ is given by

$$X(n) = Y(n) + W(n) \qquad n = 1, 2 \ldots \tag{4.1}$$

Where the noise $W(n)$ is a normal distributed random variable given by the probability density function

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

$W(n)$ is denoted

$$W \sim \mathcal{N}\left(\mu, \sigma^2\right) \tag{4.3}$$

Where the mean is μ and the variance is σ^2.

The map is a $50x50$ square map, and the possible jumps are limited to the neighboring squares, which makes the channel model reasonable to simulate using a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. The mean is assumed always to be $\mu = 0$, but the variance can deviate which makes it a parameter that will have to be investigated.

A walk with and without noise is illustrated in Figure 4.6. The red line is the walk with noise mean $\mu = 0$ and variance $\sigma^2 = 1$. The green line is the same as the red line expect for the variance of the noise which is $\sigma^2 = 2$. The blue line is the without any noise.

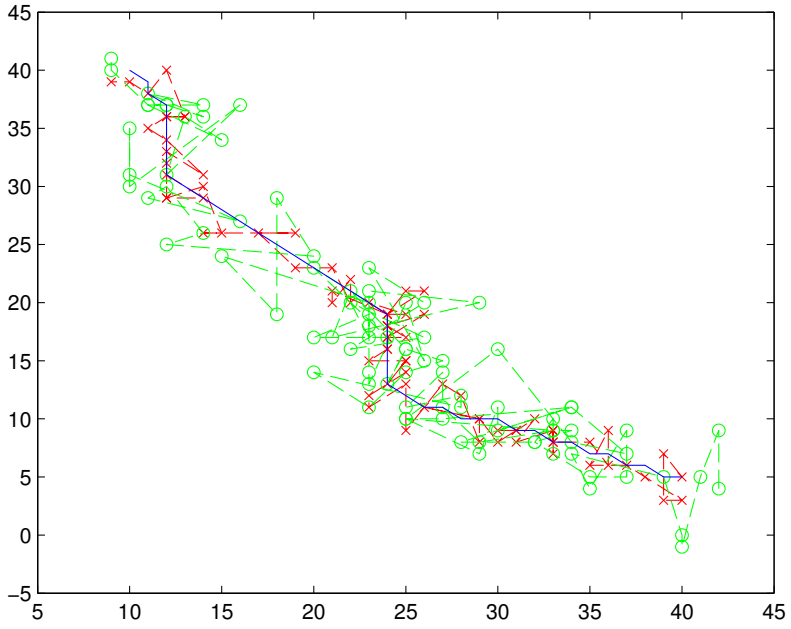

Figure 4.6: A walk with and without noise. The blue line is the without noise. The red line is the walk with noise where mean $\mu = 0$ and the variance $\sigma^2 = 1$. The green line is the walk with noise where mean $\mu = 0$ and the variance $\sigma^2 = 2$

The distance at each step is calculated from the noise to the actual walk in Figure 4.7. The green lines with green circle end markers are the euclidean distance for each step from the noise with variance $\sigma^2 = 2$ to the actual walk. The red lines with red squared end markers are the distance for each step from the noise with the variance $\sigma^2 = 1$ to the actual walk.

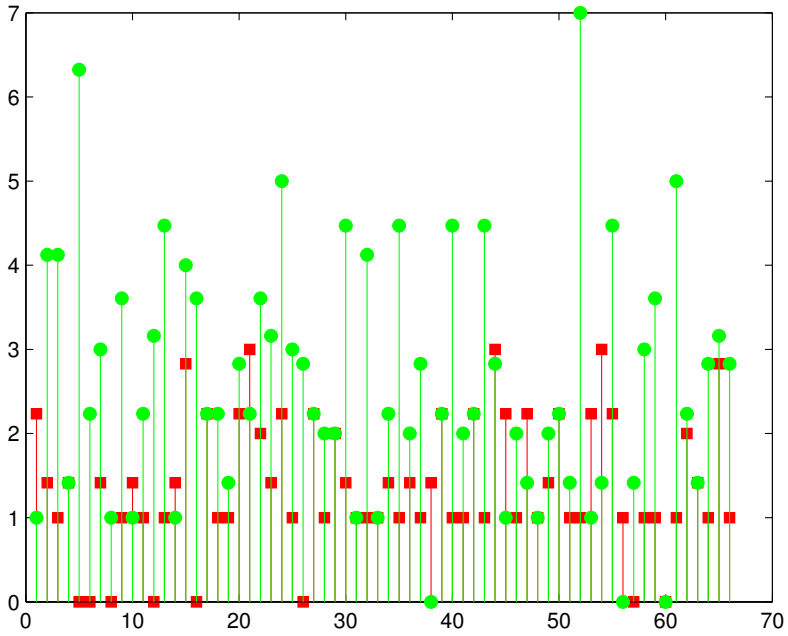

Figure 4.7: The distance from the actual walk to the noise at each step. The green lines with green circle end markers are the distance to the noise with variance $\sigma^2 = 2$ and the red lines with red squared end markers are the distance to the actual walk from the noise with variance $\sigma^2 = 1$

Chapter 5

Results

In this chapter all the results from the different tests are presented. The tests can be divided into three sections.

- Preliminary test
- Test of four different scenarios
- Pedestrian simulation

5.1 Preliminary Tests

In order to perform the larger tests some small tests has been conducted in order to find the right settings for the bigger test.

The two filters; Viterbi (filter two) and Reduced complexity Viterbi (filter three) described in Section 3.3 and Section 3.4 respectively is tested for run time performance and filter performance. This is done in Section 5.1.1.

The forgetting factor of the adaptive filter is a variable that needs to be tested in the pursuit of finding the best forgetting factor. The test is done in Section 5.1.2.

The adaptive Viterbi has two different implementation methods: the self learning adaptive Viterbi and the external adaptive Viterbi. The test is done in Section 5.1.3.

5.1.1 Filter Time Measurements

Two versions of the Viterbi algorithm has been developed. One algorithm that calculates probabilities for the entire map see Section 3.3 and one that only calculates probabilities for the part of a selected part of the map see Section 3.4 in order to reduce complexity.

The two Viterbi algorithms where tested for one path, which for both of them gave a mean euclidean distance to the actual path of 0.7251. The execution time was, however, very different. For the one utilizing the entire map the run time was 2887.8 seconds and for the limited one it was 13.73 seconds. Because these two algorithms give the same results but one of them has significantly lower runtime, it is chosen only to use the reduced complexity version in the following tests - from here on named Viterbi.

5.1.2 Different Forgetting Factors for the Adaptive Filter

The self adaptive filter is tested with three different forgetting factors 0.01 0.005 and 0.002. A sample of 500 walks on the same path is generated and is filtered with the different forgetting factors, the mean error is calculated and plotted to see if an evolution in the error can be seen.

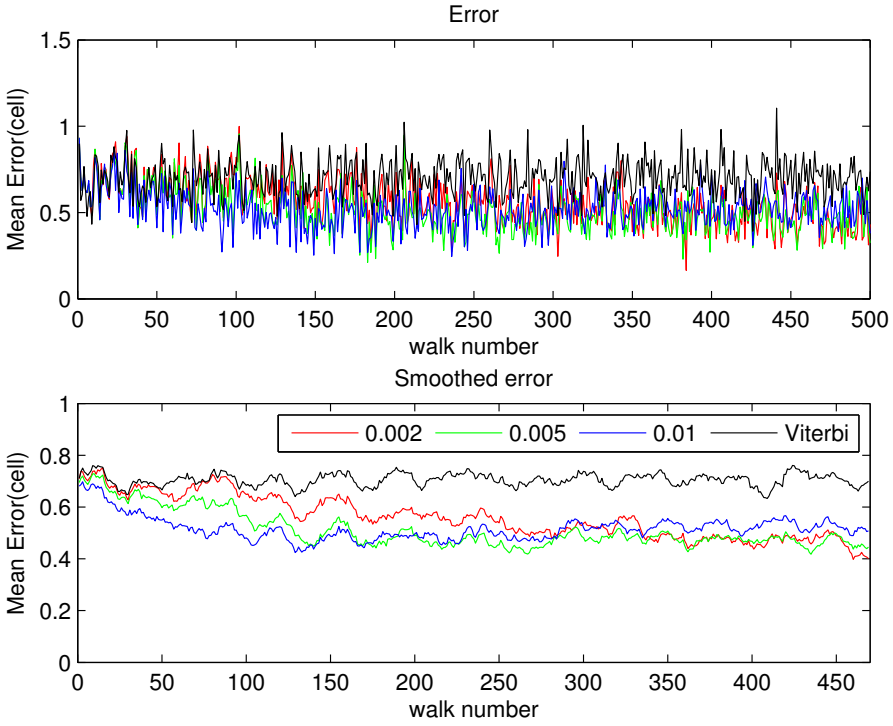

Figure 5.1: The evolution of the error in the self adaptive filter for different forgetting factors.

The evolution of the error seen in Figure 5.1 is in the first subplot very hard to conclude anything from except that it has a slight downward-slope. In order to better highlight any evolution the data is run through a smoothing filter with a window size of 15, this filter removes some of the samples in the beginning and in the end. In the smoothed plot the evolution is easier to see. The downward-slope is easily seen. Especially at the start of the test with a forgetting factor of 0.01 is used. The higher forgetting factor does not appear to converge to a higher value than the lower forgetting factors. The number of training samples and the forgetting factor are chosen based on the results from the test shown in Figure 5.1. Here on the forgetting factor is set to 0.005 and the number of training samples is 200.

5.1.3 Different Composition Filter for the Adaptive Filter

In Section 3.6 it is suggested to use to different filters input filters to insure the measurements are next to each other; one is simply just connecting the measurements to form a trajectory (simple composition), whereas the other is Viterbi.

It is easily seen from Figure 5.2 that Viterbi is a better filter than the simple composition filter. In order to find out how much better a hypothesis test is performed. The type of test performed is a paired one-sided t-test. In this type of test a new variable is made, $W_i = Mean\_simple_i - Mean\_viterbi_i$ Where $Mean\_simple_i$ is the mean error of $walk_i$ when using the simple filter for training and $Mean\_viterbi_i$ is the mean error when using Viterbi.
Mean and standard deviation of W is calculated to:

$$\bar{W} = mean(W) = 0.1016 \tag{5.1}$$

$$S_W = std(W) = 0.0889 \tag{5.2}$$

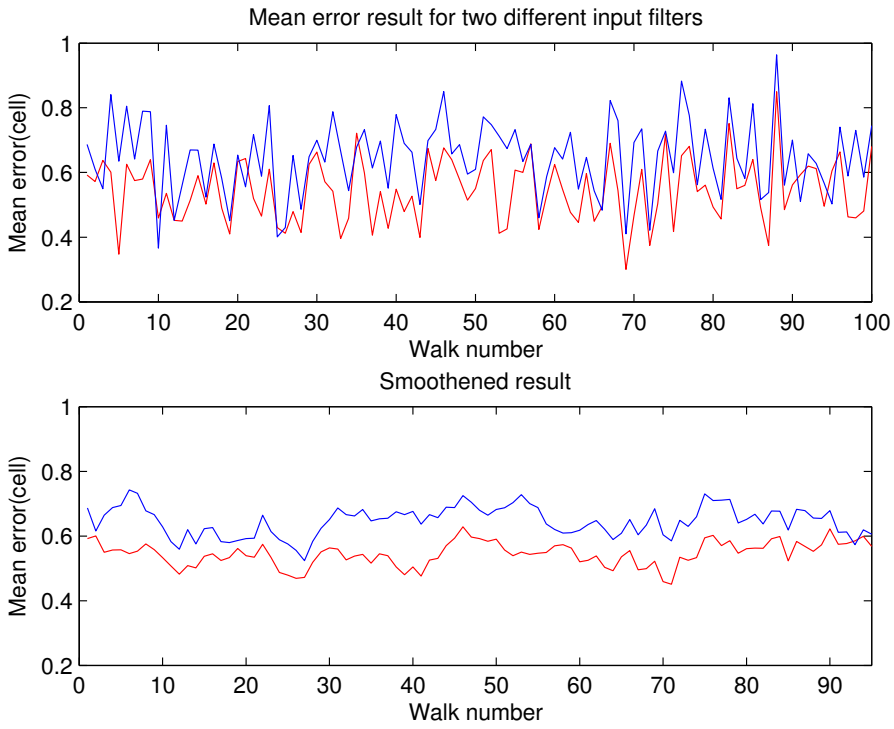

Figure 5.2: Input filter test. The upper graph shows the mean error for the 100 walks for each composition filter. The red line is for Viterbi filter. The blue line is for the simple composition filter. The lower graph is the same as the upper just smoothened with a moving average with a window size of 5.

A hypothesis is set up where μ_0 improvement.

$$H_0 : \mu(W) \leq \mu_0 \tag{5.3}$$
$$H_1 : \mu(W) > \mu_0 \tag{5.4}$$

In order to test the hypothesis Eq. (5.5) and Eq. (5.6) from [Ros04] is used. Where α is the significances level.

$$accept\ H_0\ if\ \frac{\sqrt{n} \cdot \left(\bar{W} - \mu_0\right)}{S_W} \leq t_{\alpha,n-1} \tag{5.5}$$
$$reject\ H_0\ if\ \frac{\sqrt{n} \cdot \left(\bar{W} - \mu_0\right)}{S_W} > t_{\alpha,n-1} \tag{5.6}$$

The improvement μ_0 is swept from 0 to 0.2 with a step size of 0.01 and an improvement of 0.08 is found with a 95% confidence interval. Here on the composition filter will be Viterbi

5.2 Test of Four Different Scenarios

The four scenarios described in Section 4.1 is used for testing the four different filters i.e. the moving Average, the Viterbi, the directional Viterbi and the adaptive Viterbi filter. Each test is done once for all four filters, and the mean euclidean distance from the actual path to the filtered path at each step is calculated and used comparison. The input data at each test is the same for all four filters.

The Adaptive Viterbi filter is trained with 200 paths for the specific test that all are added with noise before the test is done. The adaptive Viterbi filter is configured with a forgetting factor of 0.005 see Section 5.1.2.

A larger version of the figures can be found in Appendix C

5.2.1 Line Test

The line test is described in Section 4.1 as scenario A. The scenario is a straight line from point (10,25) to (40,25) and the input for the filters are the actual path added with noise and random velocity (see Section 4.3).

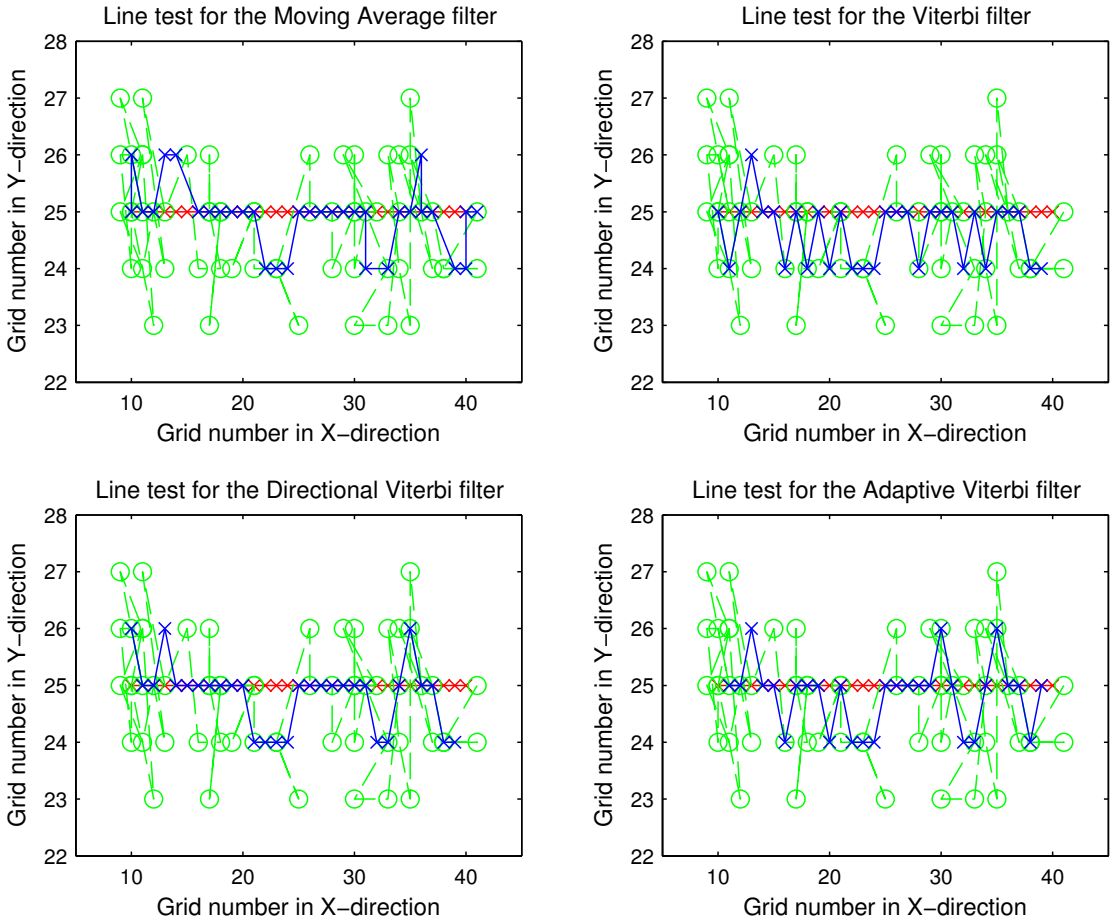

Figure 5.3: The line test for the four filters. The red line indicates the actual path. The Green indicates the path added with noise which is the input for the filters. The Blue line is the output from the different filters

On Figure 5.3 the top left plot shows the results from the Moving Average filter. The output

from the Moving Average filter is shown as the blue line. The green line is the actual walk added with noise, and the red line is the actual path. Figure 5.3 shows a small deviation in the Y-direction for the Moving Average filter, this means the Moving Average filters look to perform quite well for a straight line. However, the error in the X-direction (the velocity) is not displayed in Figure 5.3, but the error can be seen in Figure 5.4 where the Moving Average does not appear to perform that well compared to the other filters. The mean error for the Moving Average in the line test is 0.928.

The Viterbi filter results for the line test are shown on Figure 5.3. Compared with results from the Moving Average filter next to it, the Viterbi filter seems to perform worse, but according to Figure 5.4 this is not the case. The Viterbi filter has a mean error for the line test of 0.643.

The directional Viterbi filter results for the line test is also shown in Figure 5.3 this time in the bottom left corner. The directional Viterbi performs remarkable better than the Moving Average and Viterbi filter. The modification done for the directional Viterbi filter compared with the standard Viterbi proves to increase the performance for the line test. The error for the directional Viterbi can be found in Figure 5.4. The mean error for the directional Viterbi filter is 0.614.

The line test for the Adaptive Viterbi filter is performed with the external adaptive Viterbi filter trained with 200 walks before the test and with a forgetting factor of 0.005. The results are shown in Figure 5.3 and are very similar to the results from Viterbi and directional Viterbi show in the same figure. The mean error for the adaptive Viterbi filter in the line test is 0.651. The fact that the adaptive Viterbi performs worse than both Viterbi and directional Viterbi can be a result of the high fluctuation in the mean error for the adaptive Viterbi shown in Figure 5.1, a test with more sample is performed in Section 5.3.

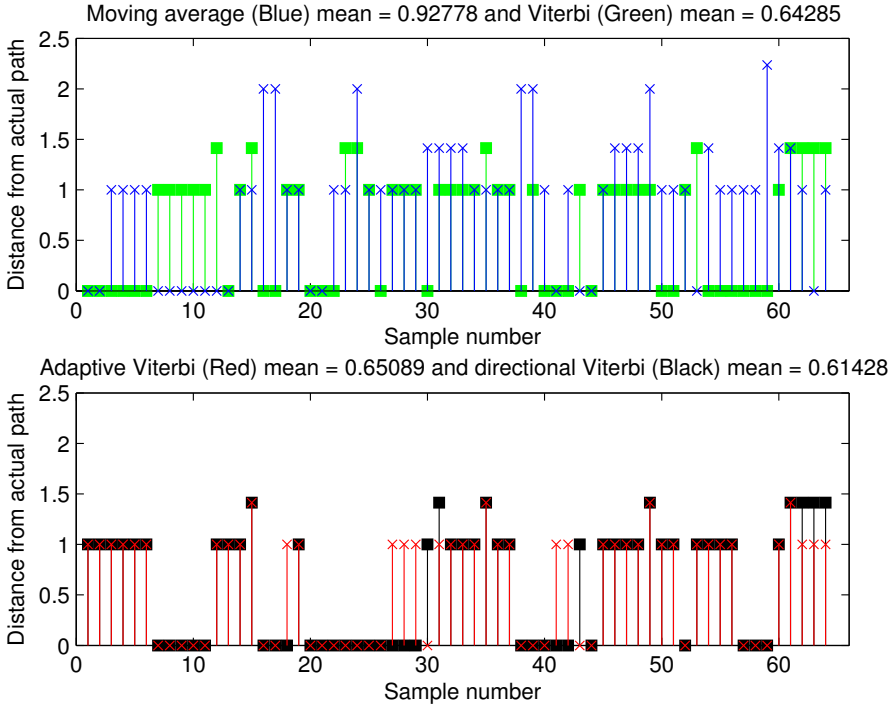

Figure 5.4: The error in the line test. The upper graph shows the error for the Moving Average (blue), and the green lines are the error for the Viterbi filter. The bottom graph shows the directional Viterbi filter (Black) and the adaptive Viterbi (Red)

The Moving Average filter has the worst performance which was expected. The Moving Average

filter does not use any information about mobility of the object. The directional Viterbi has the best performance in line test. This proves a better performance when the correct mobility model is used to filter the incoming positions. The adaptive Viterbi was excepted to perform close to the directional Viterbi. However, this was not the case. It actually performed worse than Viterbi, which was very surprising. The standard Viterbi is only using the information that the object only moves to the neighboring cells or stays in the same cell with equal probability. The adaptive Viterbi should have learned the very straight movement pattern and performed better than the standard Viterbi. The most probable explanation for this is the high fluctuation in the mean error for the adaptive filter.

5.2.2 90° Turn Test

The 90° turn test is described in Section 4.1 as scenario B. The test is conducted by connecting two straight lines from (40,25) to (40,40) and from (40,40) to (25,40). As in the line test the external adaptive Viterbi filter is trained with 200 walks added with noise and the filter's forgetting factor is set to 0.005.

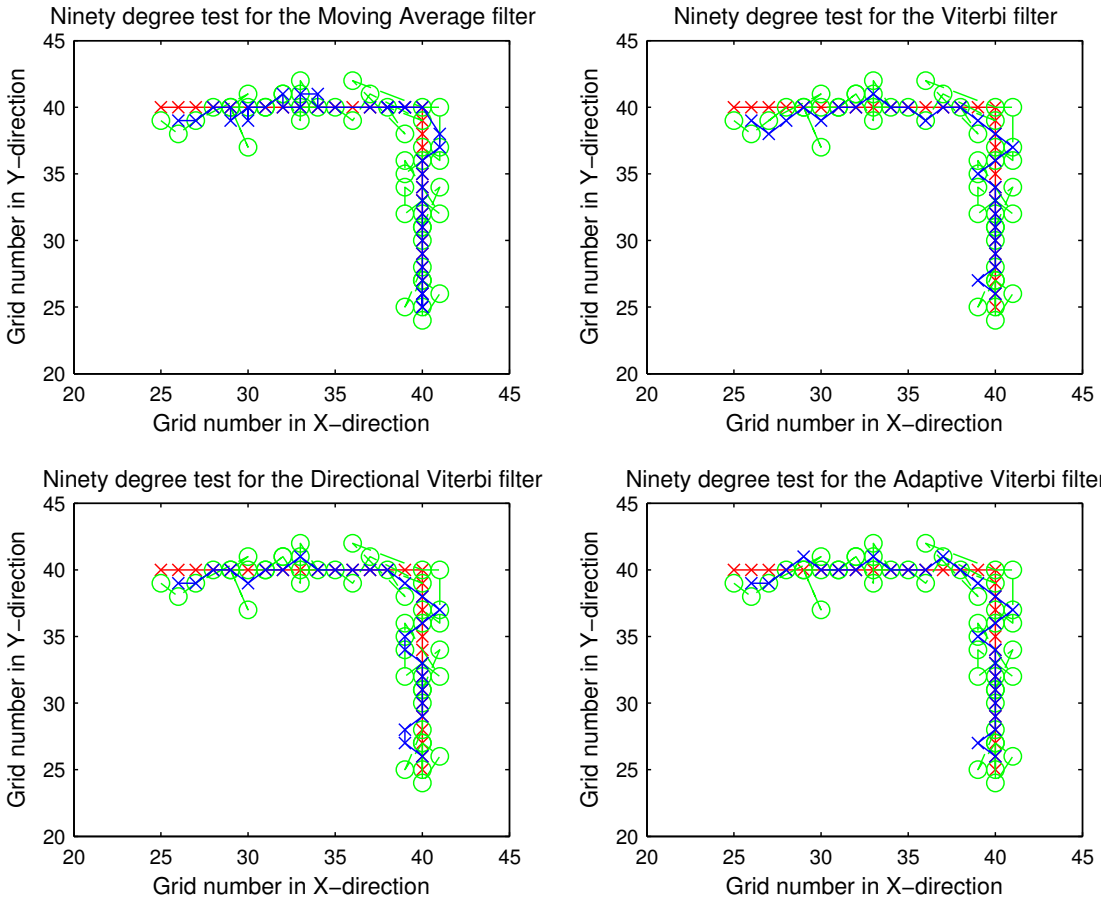

Figure 5.5: The ninety degree turn test for the four filters. The red line is the actual path. The Green is the walk added with noise which is the input for the filters. The Blue line is the output from the different filters

The 90° turn test results for the Moving Average filter is shown in Figure 5.5 in the top left corner. The results found in Figure 5.5 seems good compared with the other three filters, but as shown in Figure 5.6 the error for the Moving Average is relatively high. The Moving Average filter performs well according to Figure 5.5 because the noise is centered around the actual path, but

as Figure 5.6 states, the error in the velocity is relatively high. The mean error for the Moving Average filter in the 90° test is 1.

The Viterbi filter results for the 90° turn test are shown in Figure 5.5. The results show some deviation from the actual path where the noise diverge from the actual path. The error seen on Figure 5.6 is a little lower than the Moving Average. It can be seen that in the 90° both the Moving Average and the Viterbi filter are not performing very well. The mean error for the Viterbi filter in the 90° is 0.775.

The results for the directional Viterbi in the 90° turn test are shown in Figure 5.5 lower left corner. The results looks like the results from the standard Viterbi filter. The directional Viterbi is more willing to go in a straight line which is the difference for the two filters both in results and implementation. Looking at sample number 20 - 25 on Figure 5.6 it is easily seen where the 90° turn is. The directional Viterbi filter is not performing very well in this 90° turn. The mean error for the directional Viterbi filter in the 90° test is 0.689.

The results for the adaptive Viterbi filter for the 90° turn test is shown in Figure 5.5. The results are very similar with those from the directional Viterbi which probably is because of the resemblance of the two filters when filtering a straight line. The 90° turn test is basically two straight lines. The adaptive Viterbi filter was expected to perform better in the turn, but both Figure 5.5 and Figure 5.6 shows no significant performance improvement compared with both Viterbi and the directional Viterbi. The mean error for the adaptive Viterbi filter in the 90° turn test is 0.645.

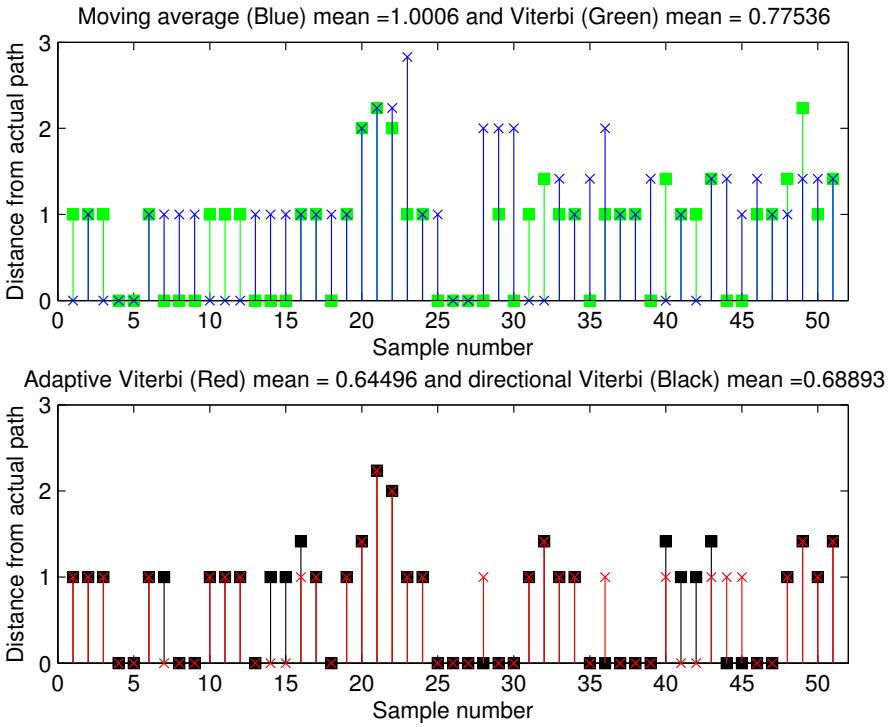

Figure 5.6: The error for the four filters in the 90° turn test. The upper graph shows the Moving Average (blue) and the Viterbi (green). The lower graph shows the directional Viterbi (black) and the adaptive Viterbi (red). In both graphs the 90° turn is at sample number 20 - 25.

On Figure 5.6 the error for all four filters are shown. The Moving Average filter is the worst

with a mean of 1 where Viterbi has 0.776, directional Viterbi has 0.689 and the adaptive Viterbi has 0.645. directional Viterbi and adaptive Viterbi is performing equally in the turn (sample number 20-25) which means that for this specific example no major improvements have been obtained using the adaptive Viterbi over the directional. The difference in the two mean values is probably due to fluctuations in the error measurements like seen in Figure 5.1.

5.2.3 180° Turn Test

The 180° turn test is described in Section 4.1 as scenario C. The test is conducted by combining two straight lines from (10,20) to (40,20) and from (40,20) to (10,20).

The adaptive filter is trained by 200 walks added with noise and configured with a forgetting factor of 0.005

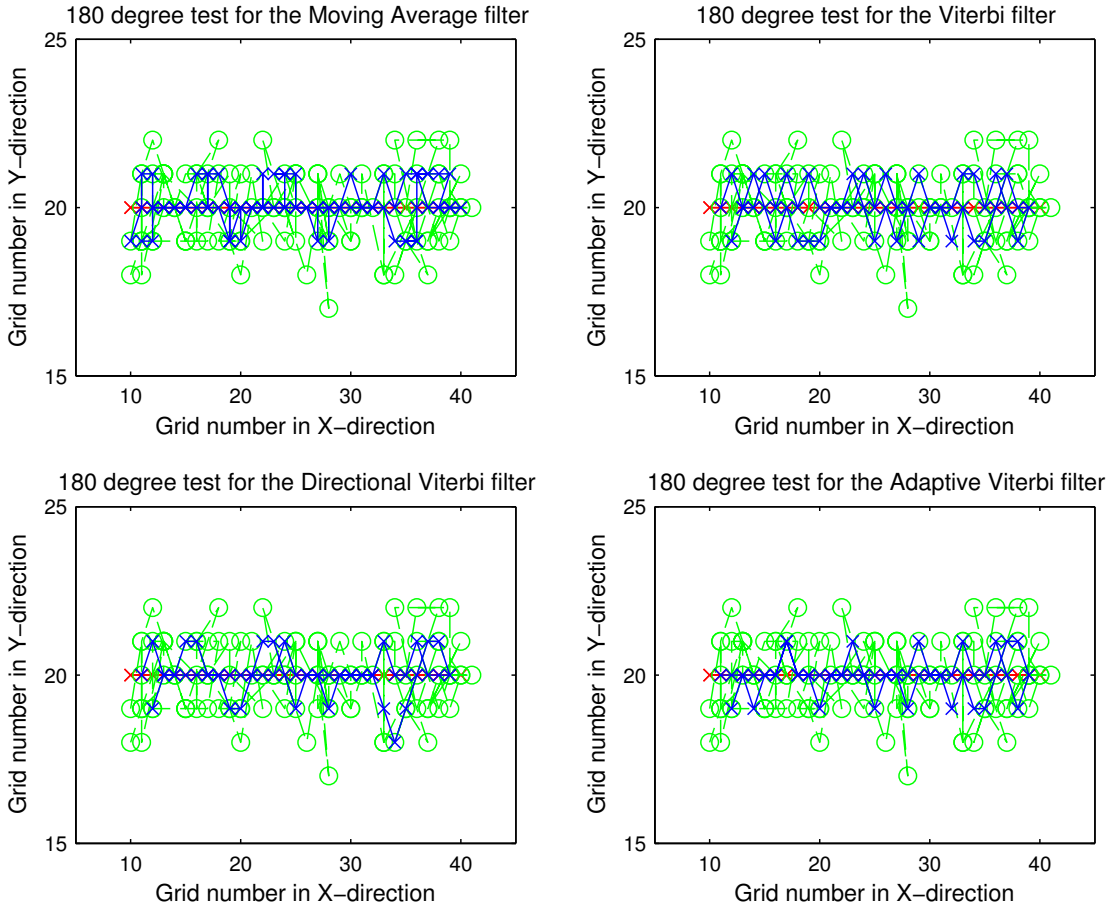

Figure 5.7: The 180° turn test for the four filters. The red line are the actual path. The Green are the walk added with noise which is the input for the filters. The Blue line is the output from the diffrent filters

The Moving Average filter results from the 180° turn test is shown in Figure 5.7. Judged from Figure 5.7 the Moving Average filter performs very well compared to Viterbi, directional Viterbi and adaptive Viterbi, but according to Figure 5.8 the Moving Average is not very good. The mean error for the Moving Average filter in the 180° turn test is 0.99.

The Viterbi filter results are shown in Figure 5.7. The Viterbi filter performs better than the Moving Average filter. The turn can be seen in Figure 5.8 in sample number 60-70. The Viterbi

filter does not perform particular different when turning 180°. The mean error for the Viterbi filter in the 180° turn test is 0.7.

The directional Viterbi filter for the 180° turn test results are found on Figure 5.7 in the bottom left. The directional Viterbi filter performs slightly better than the standard Viterbi filter with a mean error of 0.683 which is approximately 0.017 better. Such a small difference might just be a result of the fluctuation in error measurements.

The adaptive filter results are shown in Figure 5.7. The results are significantly better than for all the other filters. In Figure 5.8 the adaptive Viterbi filter is compared with the directional Viterbi. The figure states a clear improvement for the adaptive Viterbi filter. The mean error for the adaptive Viterbi filter in the 180° turn test is 0.584.

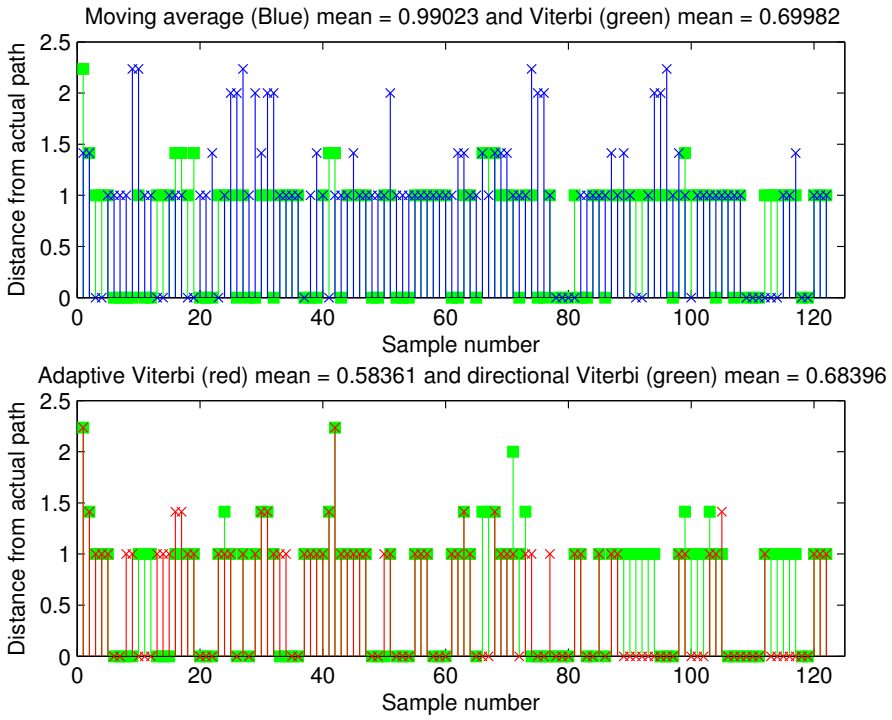

Figure 5.8: The error for the four filters in the 180° turn test. The upper graph shows the Moving Average (blue) and the Viterbi (green). The lower graph shows the directional Viterbi (black) and the adaptive Viterbi (red). On both graphs the 180° turn is at sample number 65 - 75.

The error for the four filters in the 180° turn test is shown on Figure 5.8. The 180° turn is located at sample number 65 - 75. The adaptive Viterbi is performing slightly better than the other filters in the turn. The adaptive Viterbi performance is significantly better than the other filters, but like the other test this might just be a result of fluctuation.

5.2.4 Soft Turn Test

The soft turn test is described in Section 4.1 as scenario D. The test is conducted using a circle divided in cells. Like the other tests the adaptive filter is trained using 200 walks and configured with a forgetting factor of 0.005.

The Moving Average filter results can be seen in Figure 5.9. The result seems good, and it looks like the Moving Average filter is working very well for a soft turn. However, looking at Figure 5.10 it becomes clear that the filter has a high number of errors. The mean error for the Moving Average filter in the soft turn test is 0.917.

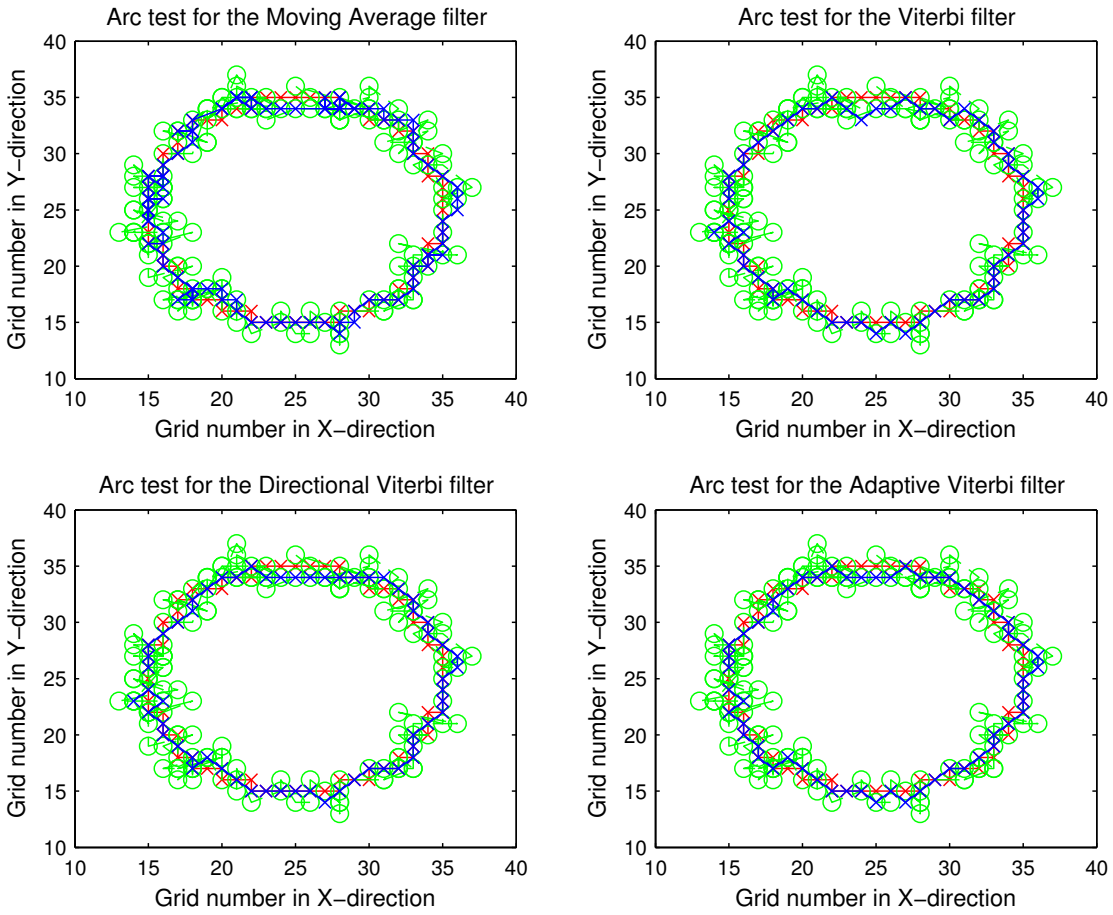

Figure 5.9: The soft turn test for the four filters. The red line is the actual path. The green line is the walk added with noise which is the input for the filters. The Blue line is the output from the different filters

The results from the Viterbi filter in the soft turn test is shown in Figure 5.9 upper right corner. The results seems very good which also is the case for Figure 5.10. The mean error for the Viterbi filter in the soft turn test is 0.598, which is very low.

The directional Viterbi filter result for the soft turn test can be found in Figure 5.9 lower left corner. The directional Viterbi filter is not performing that good, it is clear that the filter in this test seems to be going a little to much straight which is why it performs worse than the standard Viterbi and the adaptive Viterbi filter. The mean error for the directional Viterbi filter in the soft turn test is 0.623.

The results for the adaptive Viterbi filter can be found in Figure 5.9. The adaptive Viterbi filter is the best filter for the soft turn test which can be seen in Figure 5.10. The mean error for the adaptive Viterbi in the soft turn test is 0.589.

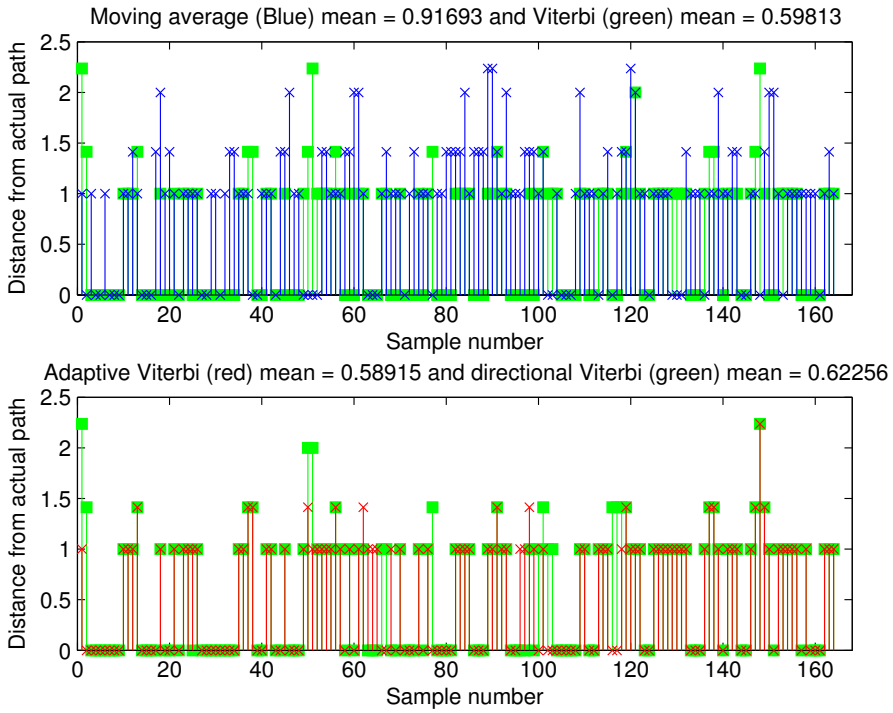

Figure 5.10: The error for the filters in the soft turn test. The upper graph shows the Moving Average (blue) and the Viterbi (green). The lower graph shows the directional Viterbi (black) and the adaptive Viterbi (red)

On Figure 5.10 the error for all the filters can be found. The worst performance is by the Moving Average which has some outliers compared with the other filters. The directional Viterbi has the second worst performance which makes sense due to the fact that the directional Viterbi filter is optimized to perform better in a straight line. The adaptive Viterbi and the standard Viterbi is performing close to equally.

It is hard to conclude munch from these test as only one walk is used in this test, but the expected problems for the directional viterbi to turn appears to be correct. In order to test the performance of the different filters more walks needs to be run though.

5.3 Final test

A final test of the filters on all the 34 paths from Section 4.2 is performed. The four filters chosen for this test is the moving average, Viterbi , directional Viterbi and external adaptive Viterbi. The reason the self adaptive Viterbi is left out is because it has to sweep though a big amount of data for training, whereas the external adaptive Viterbi can be trained significantly faster due to the lower complexity of normal Viterbi.

The test is set up by running 10 walks of every path though all four filters, the external adaptive filter is trained with 300 samples first, a forgetting factor of 0.005 is used. Two parameters are saved for further analysis; the filtered walk and position the filter deems most likely for every observation. This will be referred to as off-line and on-line test parameters.

5.3.1 Preliminary Results

The mean error is calculated for all four filters, and both test parameters

Filter	Error off-line	Error on-line
Moving average	1.12	1.12
Viterbi	0.67	1.13
Directive Viterbi	0.67	0.99
Adaptive Viterbi	0.56	0.78

Table 5.1: Mean Errors for the different filters

The mean error for both parameters is the same for the moving average filter because it does not use any information about the following measurements. It was expected that the Viterbi was significantly worse in the case of the on-line parameter because it has difficulties with the ends of the walks. Also the filters as expected preform better off-line that on-line due to more information being available.

5.3.2 Hypothesis Testing

The mean errors is a good preliminary result but able to say something about the actual difference in performance between two filters a hypothesis tests is performed. The type of hypothesis test used is the paired one-sided t-test. A new variable W is created this is difference in mean error of the two filters $X_i - Y_i = W_i$ where X_i is the means error of the i'th walk of one filter and X_i is the other. The W variable is then tested for mean. Because a positive performance is expected the variable μ_0 that represent the improvement is introduced and swept from 0 to 0.8 with a step size of 0.01, and α the significance level set to 95%.

$$H_0 : \mu(W) \leq \mu_0 \tag{5.7}$$
$$H_1 : \mu(W) > \mu_0 \tag{5.8}$$

In order to test the hypothesis Eq. (5.9) and Eq. (5.10) from [Ros04] is used.

$$accept\ H_0\ if\ \frac{\sqrt{n} \cdot (W - \mu_0)}{s_W} \leq t_{\alpha,n-1} \tag{5.9}$$
$$reject\ H_0\ if\ \frac{\sqrt{n} \cdot (W - \mu_0)}{s_W} > t_{\alpha,n-1} \tag{5.10}$$

n is the number of samples and s_W is the sample standard deviation.
The performance off-line of the filters is tested against each other.

Filter	Viterbi	Directive Viterbi	Adaptive Viterbi
Moving average	0.43	0.43	0.54
Viterbi	-	-	0.09
Directive Viterbi	-	-	0.10

Table 5.2: The improvementμ_0 where the hypothesis is still accepted

Filter	Viterbi	Directive Viterbi	Adaptive Viterbi
Moving average	-	0.12	0.32
Viterbi	-	0.13	0.33
Directive Viterbi	-	-	0.19

Table 5.3: The improvementμ_0 where the hypothesis is still accepted

The performance on-line of the filters is tested against each other.

The Adaptive filter is the best in all cases especially in the on-line case it preforms significantly better. An improvement over directional Viterbi of 15% for the off-line case and 19% for the on-line case is found.

5.4 Noise test

Furthermore a test on the self adaptive filters is performed in which different variance in the noise is used see Figure 5.11. In order to see if variance has any influence on the evolution of the error. Because of the filter uses feedback it might become unstable with high levels of noise, a forgetting factor of 0.005 is used.

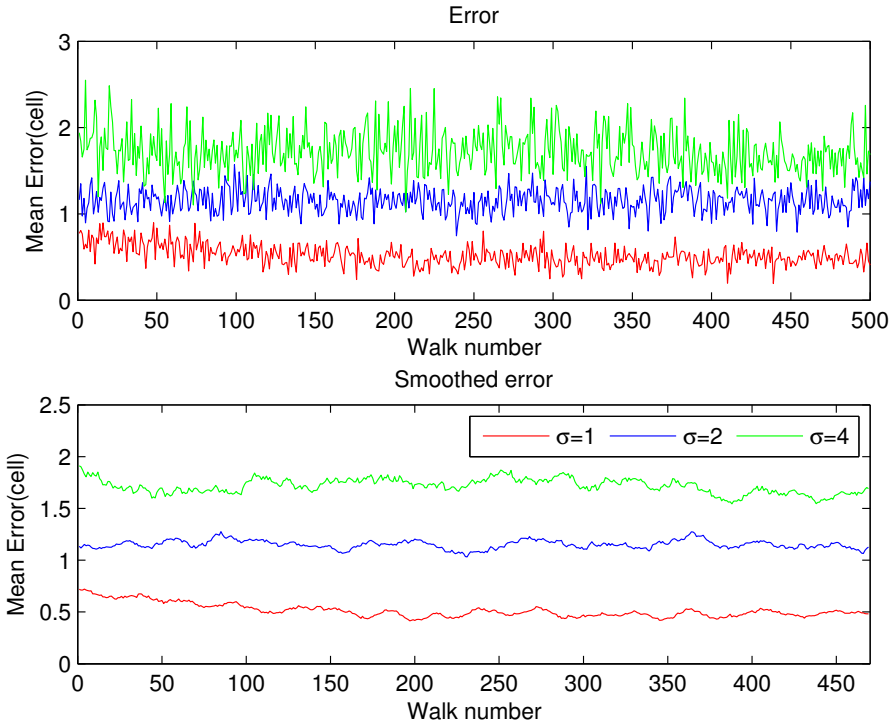

Figure 5.11: The evolution of the error in the self adaptive filter for different variances.

For $\sigma = 1$ and $\sigma = 4$ there appears to be a drop in the start but not for $\sigma = 2$ which is a bit odd and might require further investigation. This indicates that more functionalities than a simple feedback loop is present.

Chapter 6

Closure

6.1 Conclusion

The main purpose of this project was to investigate how filtering or smoothing of position measurements can improved the accuracy of the positions, by using knowledge about the mobility of the user and through the use of a self-learning geographical dependent mobility model.

Through the proposal, different filters have been proposed to reduce noisy position measurements. A Moving Average filter was proposed as a simple way of smoothing noisy position measurements. Then the use of HMMs to model the behavior of a user. This lead to using knowledge about the mobility of a pedestrian which changed the model to enhance straight trajectories. Finally this lead to changing the mobility model so that it adapts to different user behavior geographically. This means that the last proposed model was able to learn a user behavior by using training data. All the filters, except the Moving Average, used Viterbi to find the most probable path.

All the filters was tested on four simple scenarios which was believed to be the components in a pedestrians trajectory. The simulations was done by taking the actual paths and add them with Gaussian noise. The results from the test showed the Adaptive Viterbi filter performed best in all test except for one test which was a straight line test where the Directional Viterbi performed best.

The filters was also tested in a simulation of a pedestrian where a path search algorithm was used to generate actual paths which then was added with noise. The test had 34 different paths which all was filtered 10 times. The adaptive Viterbi filter was trained with 300 measurements for each 34 paths. The mean error (euclidean distance to the actual path at each step) for each filter was found for both the one step (on-line) and in the finished path (off-line). The mean error is calculated for all filters and shown in Table 6.1, this shows an improvement for the adaptive Viterbi filter.

Filter	Error off-line	Error on-line
Moving average	1.12	1.12
Viterbi	0.67	1.13
Directive Viterbi	0.67	0.99
Adaptive Viterbi	0.56	0.78

Table 6.1: Mean Errors for the different filters in the pedestrian test

Based on the results from the pedestrian simulation a paired one-sided t-test of means was carried out to test all the filters against each other. The test shows an improvement of 0.1 for the adaptive Viterbi over the directional Viterbi in the off-line test. The result from the on-line test was 0.19, which corresponds to an improvement of for the off-line test of 15% and for the on-line test 19%.

The last test done was a noise test different variance levels in the noise was tested with the adaptive Viterbi filter. The test was done to investigate if the filter would become unstable if the variance of the noise increases. The test did not reveal any significant signs of instability. However, the test revealed some odd drops in the mean error which for the future should be investigated.

6.2 Future Work

There is still some work to be done before a actual implementation is ready, a further investigation with different standard deviations and the effects on the performances of the filters is needed. The odd results found in Section 5.4 indicates that there might be a problem which will have to investigated further.

The two versions of the adaptive Viterbi filter, the external updating adaptive Viterbi and the self updating adaptive Viterbi, has not been test thoroughly to see which one of them performs the best in all situations. This will have to be investigated further.

An other topic that has to be further investigated is runtime optimization of the adaptive filter. It should be possible to use the same technique as for Viterbi to remove the last dependency of the complexity on the map size, it is only an implementation problem.

Also since most positioning system uses a particle filter and the fact that a version of the particle filter that uses a directional model already has been made [RPN+08] a version using a geographical adaptive mobility model would be a good next logical step.

Appendix A

$A^\star$

In order of finding the shortest path from a node A to goal node B, an algorithm called $A^\star$ can be used. $A^\star$ is a "best-first" search algorithm which utilizes the relative distance to find the shortest path. The "best-first" search rely on a sum of the relative distance and the cost of taking the current route from the start node to the current node. The distance from the current node to the goal node is calculated by some distance measure for example Manhattan distance or euclidean distance [Les05]. The cost of taking the current route is denoted $g(x)$ and the distance to the goal node is denoted $h(x)$. The distance plus cost sum is denoted $f(x) = g(x) + h(x)$.

A.1 The process

The flowchart on Figure A.1 illustrates how the $A^\star$ algorithm works. The algorithm starts by setting some initial values for *closedset*, *openset*, $came\_from$, $g(x)$, $h(x)$ and $f(x)$. The values can be seen in Figure A.1

closedset is the nodes that has been visited.
openset is the nodes a possible path has been found to but has yet visited.

Where as: $g(x)$, $h(x)$ and $f(x)$ is values calculated for every node in *openset*, and $came\_from$ is set in all to previous node in all *openset* nodes. The Function neighbors return the neighbors of the node it is called with.

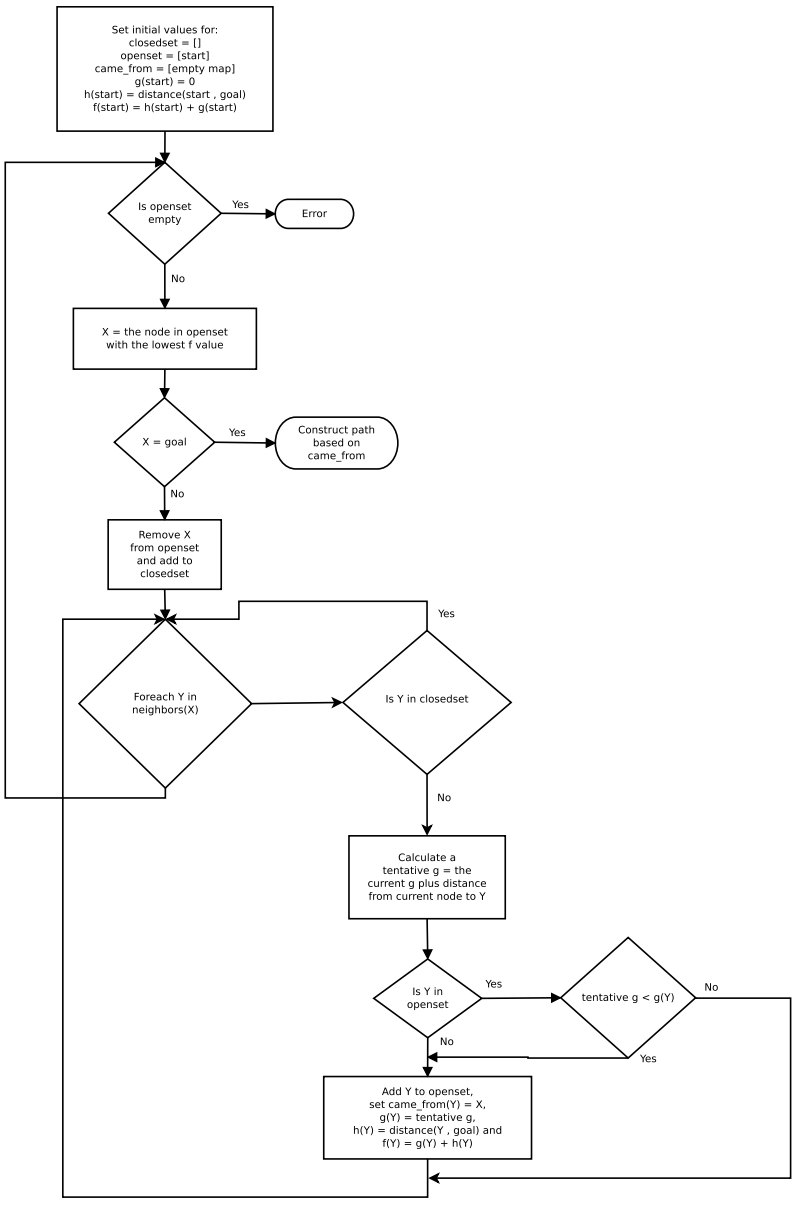

Figure A.1: Flowchart for the $A^\star$ algorithm

Appendix B

Viterbi

In order to do the filtering of the HMM's the viterbi algorithm is used, a simple introduction to the functionality of the viterbi algorithm is provided in this appendix.

B.1 Hidden Markov Models

A HMM is a Markov model where it is impossible to observe the state of the Markov model directly instead it is possible to observe a set of emissions from the HMM, in many system the states of the Markov model and the emissions is the same, especially in communication systems, this makes a bad example because both internal states and emissions have the same name but is not the same. Instead an example with to internal states rainy and sunny, in order to tell the weather and 3 emissions states run, swim, gym. This HMM is show i Figure B.1
The different statesS_i and emissionsE_j

- State 1 sunny
- State 2 rainy
- Emission 1 run
- Emission 2 swim
- Emission 3 gym

In this example the following start probability's π_i is used.

- Start Sunny $\pi_1 = 0.6$
- Start Rainy $\pi_2 = 0.4$

The used state transition probabilities a_{ij} is

- Sunny to rainy $a_{12} = 0.3$
- Sunny to sunny $a_{11} = 0.7$
- Rainy to rainy $a_{22} = 0.6$
- Rainy to sunny $a_{21} = 0.4$

Also emission probability's $b_j(k)$ is needed in order to have a HMM

- Sunny gives Run $b_1(1) = 0.6$
- Sunny gives Swim $b_1(2) = 0.3$
- Sunny gives Gym $b_1(3) = 0.1$
- Rainy gives Run $b_2(1) = 0.1$
- Rainy gives Swim $b_2(2) = 0.4$
- Rainy gives Gym $b_2(3) = 0.5$

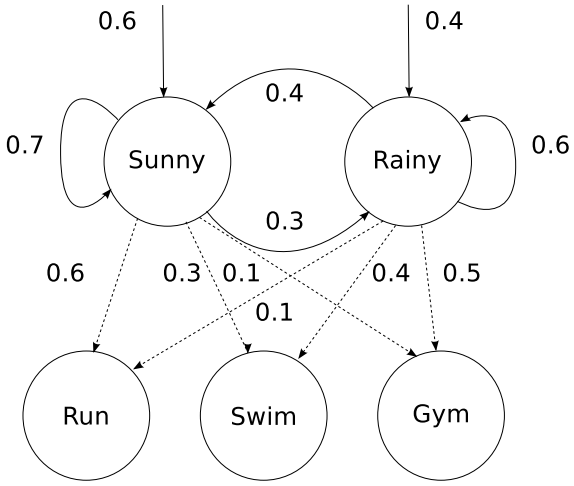

Figure B.1: Example of a Hidden Markov Model

The output from this HMM could be (run,gym,swim) all of the different combination of sunny and rainy can give this output but some is more likely that others.

B.2 Viterbi

When the output of a HMM is used to determine the states of the Markov model the Viterbi algorithm is used. In order to do this the viterbi algorithm generates something called the viterbi trellis.
In this example calculation of the viterbi trellis if the output from the HMM is Run, Swim, Gym, Run $E_1 = 1\ E_2 = 2\ E_3 = 3\ E_4 = 1$.

The first emission is run or emission 1 to calculated the probability of being in the different states this is calculated from π_i and $b_j(k)$. In the first case i and j is the same number.

State 1:

$$P(S_1 = 1|E_1) = \pi_1 \cdot b_1(1) \tag{B.1}$$
$$= 0.6 \cdot 0.6 = 0.36 \tag{B.2}$$

State 2:

$$P(S_1 = 2|E_1) = \pi_2 \cdot b_2(1) \tag{B.3}$$
$$= 0.4 \cdot 0.1 = 0.04 \tag{B.4}$$

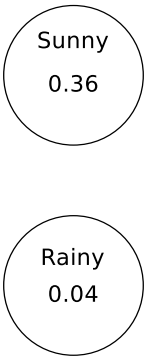

Figure B.2: First step of the viterbi algorithm trellis

The first step of the viterbi trellis is illistrated in Figure B.2.

The Emission in the second step is Swim $E_2 = 2$. Since the probabilities now shift to being the dependent on the previous state, the transition probabilities a_{ij} is now used. Since only the most probable path is interesting the maximum of $P(S_1 = i, S_2 = j|E_1, E_2)$ needs to be found. This can require many calculations depending on the amount of states.

State 1:

$$P(S_1 = 1, S_2 = 1|E_1, E_2) = P(S_1 = 1|E_1) \cdot a_{11} \cdot b_1(2) \tag{B.5}$$
$$= 0.36 \cdot 0.7 \cdot 0.3 = 0.0756 \tag{B.6}$$

$$P(S_1 = 2, S_2 = 1|E_1, E_2) = P(S_1 = 2|E_1) \cdot a_{21} \cdot b_1(2) \tag{B.7}$$
$$= 0.04 \cdot 0.4 \cdot 0.3 = 0.0048 \tag{B.8}$$

State 2:

$$P(S_1 = 1, S_2 = 2|E_1, E_2) = P(S_1 = 1|E_1) \cdot a_{12} \cdot b_2(2) \tag{B.9}$$
$$= 0.36 \cdot 0.3 \cdot 0.4 = 0.0432 \tag{B.10}$$

$$P(S_1 = 2, S_2 = 2|E_1, E_2) = P(S_1 = 2|E_1) \cdot a_{22} \cdot b_2(2) \tag{B.11}$$
$$= 0.04 \cdot 0.6 \cdot 0.4 = 0.0096 \tag{B.12}$$

In the next state the previous states probabilities is needed so the maximums for each state is saved with the information for where they came from see Figure B.3.

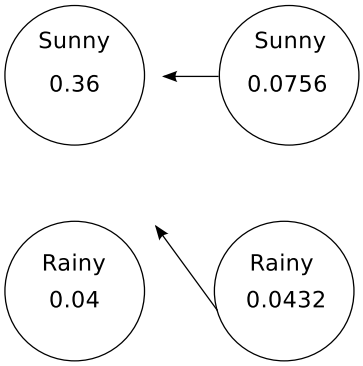

Figure B.3: Second step of the viterbi algorithm trellis

In the third step the procedure from the second step is repeated.
State 1:

$$P(S_2 = 1, S_3 = 1|E_1, E_2, E_3) = P(S_1 = 1, S_2 = 1|E_1, E_2) \cdot a_{11} \cdot b_1\,(3) \tag{B.13}$$
$$= 0.0756 \cdot 0.7 \cdot 0.1 = 0.005292 \tag{B.14}$$

$$P(S_2 = 2, S_3 = 1|E_1, E_2, E_3) = P(S_1 = 1, S_2 = 2|E_1, E_2) \cdot a_{21} \cdot b_1\,(3) \tag{B.15}$$
$$= 0.0432 \cdot 0.4 \cdot 0.1 = 0.001728 \tag{B.16}$$

State 2:

$$P(S_2 = 1, S_3 = 2|E_1, E_2, E_3) = P(S_1 = 1, S_2 = 1|E_1, E_2) \cdot a_{12} \cdot b_2\,(3) \tag{B.17}$$
$$= 0.0756 \cdot 0.3 \cdot 0.5 = 0.0756 \tag{B.18}$$

$$P(S_2 = 2, S_3 = 2|E_1, E_2, E_3) = P(S_1 = 1, S_2 = 2|E_1, E_2) \cdot a_{22} \cdot b_2\,(3) \tag{B.19}$$
$$= 0.0432 \cdot 0.6 \cdot 0.5 = 0.001728 \tag{B.20}$$

The trellis is updated with the third step, see Figure B.4.

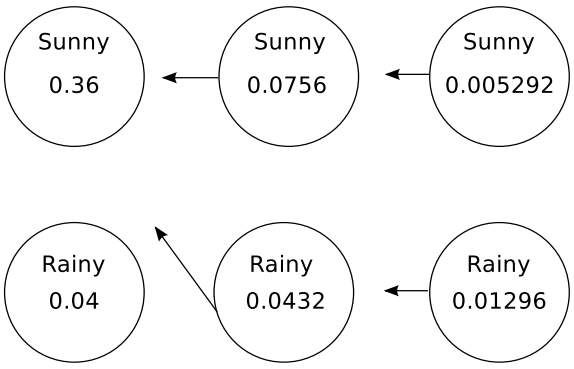

Figure B.4: third step of the viterbi algorithm trellis

In the fourth step only the same procedure is used and the resulting trellis is seen in Figure B.5

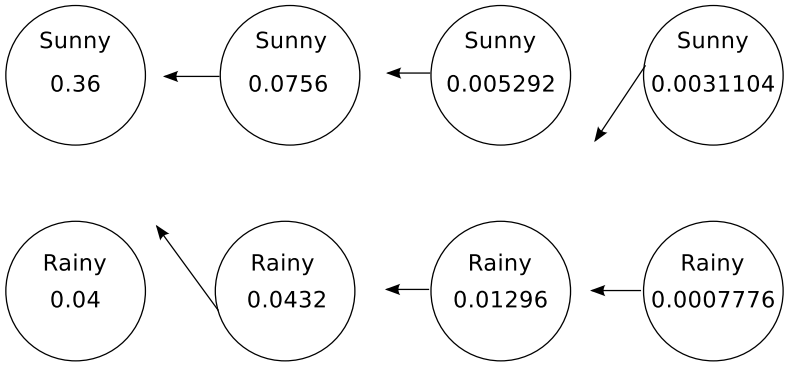

Figure B.5: Example of viterbi algorithm trellis

When the end of the viterbi trellis has been reached the node with the highest value is chosen and used to find the most likely path. This would give the result sunny, rainy, rainy, sunny as the states of the HMM.

If the viterbi trellis is long the resulting numbers can become very small and this gives a problem on many computer systems, but since it is relationship between the different probability's and not the value of them that is interesting this is solvable by using either logarithmic or timing all the probability's with the same value. The algorithm when written is basically an initialize-phase, a main-phase, and a terminate-phase.

B.2.1 Initialize Phase

The initialize phase calculates the probability of being in the different states the from the first observation using the begin probability's and the emission probability's see Algorithm 8, this is also the first step in the viterbi trellis and the path.

for *y in states* **do**
  trellis[1][y] $=\pi_y \cdot b_y\,(obs[0])$
  path[1][y] = y
end

Algorithm 8: The initialize phase of the viterbi algorithm

B.2.2 Main Phase

The main phase consist of 2 for loop's inside each other one that sweeps though the observations, and an other to move though all the states. For all the states the probability of being in that state has to be found, this is the maximum probability of all previous states timed with the transition probability's and the emissions probability. It is in the main-phase where the complexity of $O(T * |Y|^2)$ comes where T is the length of the observation sequence and Y is the number of states.

for *t in range(2,length(obs))* **do**
  for *y in states* **do**
    (prob,state) = max(trellis[t-1][x]$\cdot a_{xy} \cdot b_y\,(obs[t])$,x) for x in states
    trellis[t][y] =prob
    path[t][y] = state
  end
end

Algorithm 9: The main phase of the viterbi algorithm

The maximum probability of all states is saved and so is the state that lead to this maximum in order to be capable to backtrack the path.

B.2.3 Terminate Phase

Takes the highest end value and move backwards though the trellis in order to find the most probably path.

```
state = max(trellis[length(obs)][y],y) for y in states
walk[length(obs)] = state
for t in range(1,length(obs)-1) do
    walk[length(obs)-t]=path[length(obs)-t+1][state]
    state= walk[length(obs)-t]
end
return walk
```

Algorithm 10: The terminate phase of the viterbi algorithm

The returned walk is the walk that most probable gave the sequence of observations in the case of high noise levels it will probably not be the true path but closer to the true path than before.

Appendix C

Results From Four Different Scenarios test

This appendix contain the results from Section 5.2.

C.1 Line test

The line test is described in Section 4.1 as scenario A. The scenario is a straight line from point (10,25) to (40,25) and the input for the filters are the actual path added with noise and random velocity (see Section 4.3).

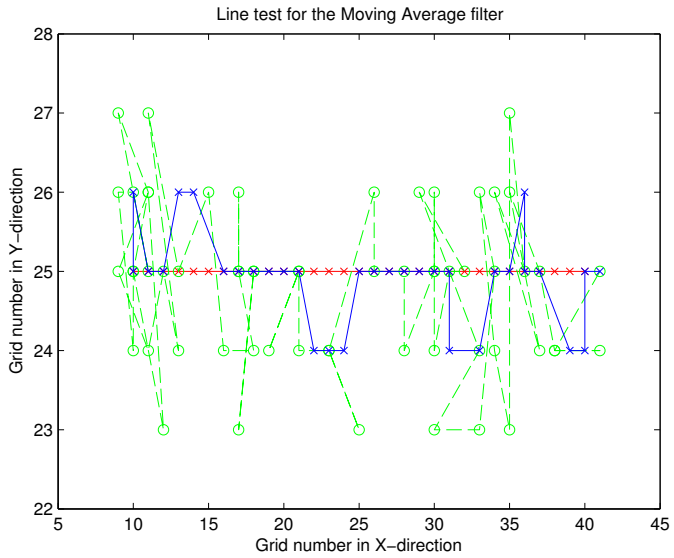

Figure C.1: The line test for the Moving Average filter. The red line are the actual path. The Green are the path added with noise which is the input for the filter. The Blue line is the output from the Moving Average filter

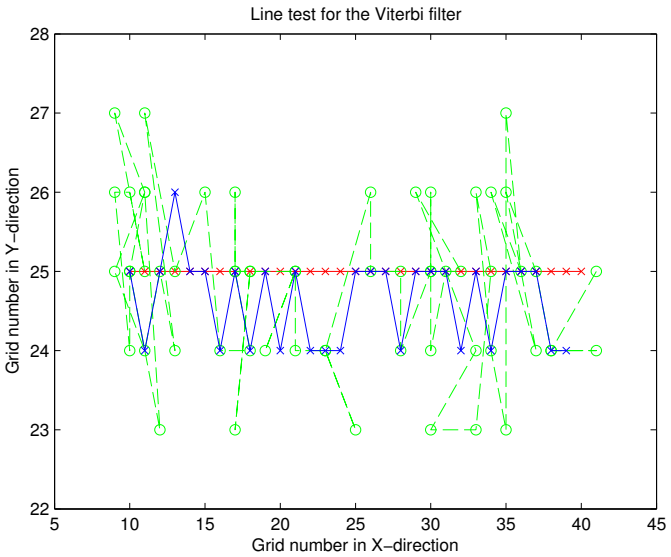

Figure C.2: The line test result for the Viterbi filter (blue). The green line is the input for the filter which is the actual path (red) added with noise

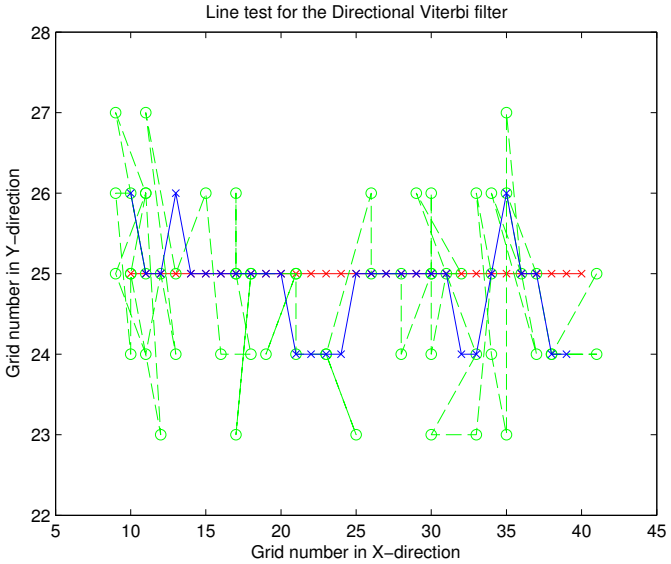

Figure C.3: The results for the directional Viterbi filter in the line test. The green line is the input which is the actual path (red) added with noise. The blue line is the output for the filter

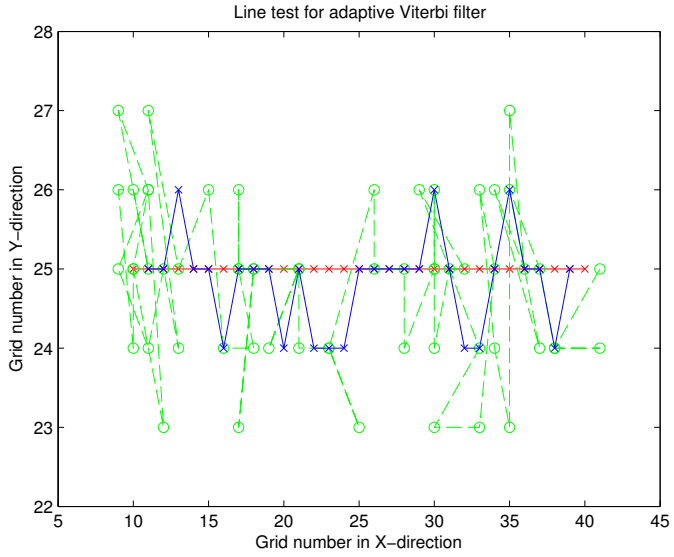

Figure C.4: The results for the line test of the adaptive Viterbi. The green line is the path added with noise (the input for the filter). The red line is the actual path. The blue line is the output for the filter

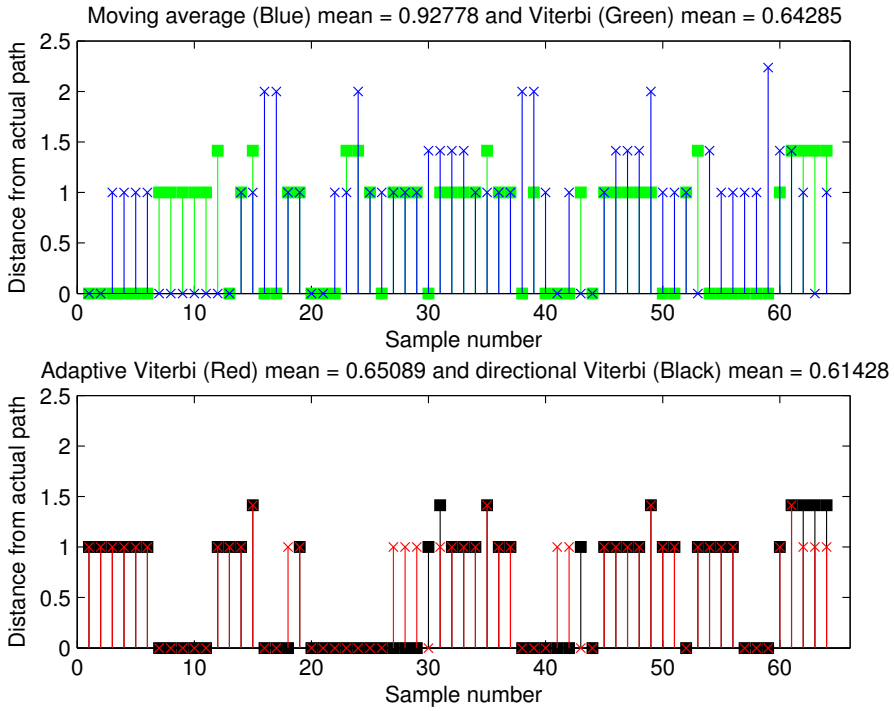

Figure C.5: The error in the line test. The upper graph shows the error for the Moving Average (blue), and the green lines are the error for the Viterbi filter. The bottom graph shows the directional Viterbi filter (Black) and the adaptive Viterbi (Red)

C.2 90° Turn Test

The 90° turn test is described in Section 4.1 as scenario B. The test is conducted by connecting two straight lines from (40,25) to (40,40) and from (40,40) to (25,40). As in the line test the external adaptive Viterbi filter is trained with 200 walks added with noise and the filter's forgetting factor is set to 0.005.

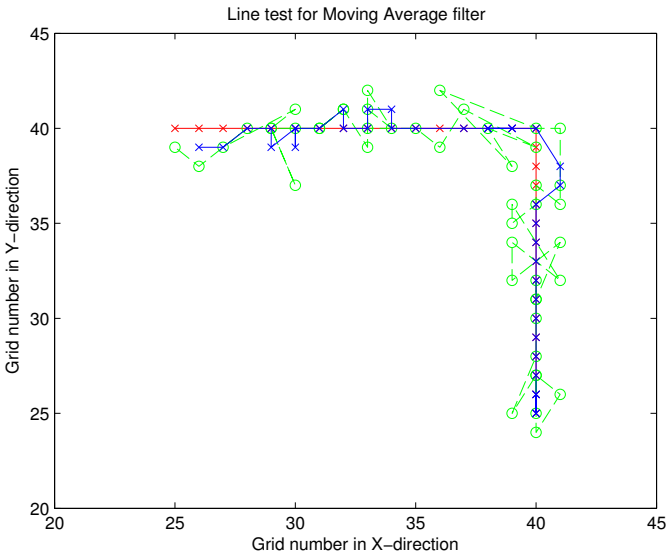

Figure C.6: The results for the Moving Average filter. The green line is the actual path (red) added with noise which is the input for the filter. The blue line is the output for Moving Average

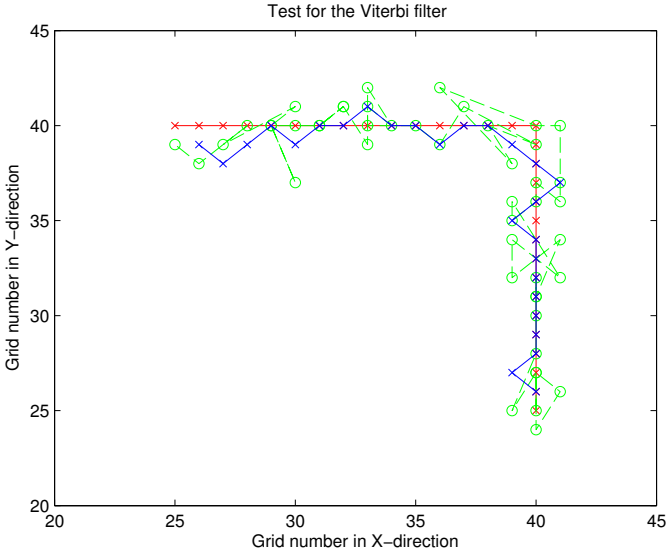

Figure C.7: The results for the Viterbi filter on the 90° turn test. The blue line is the output from the filter. The red line is the actual path. The green line is the actual path added with noise which is the input for the filter

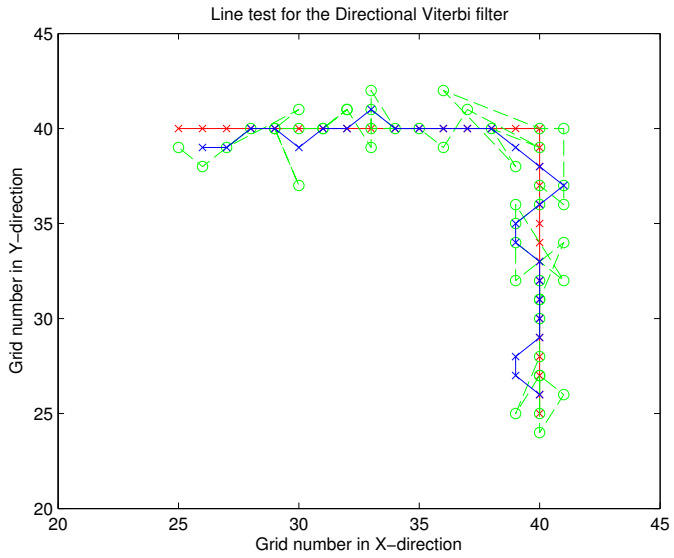

Figure C.8: The results for the directional Viterbi filter in the 90°. The blue line is the output for the filter where the green line is the input which is the actual path, the red line, added with noise

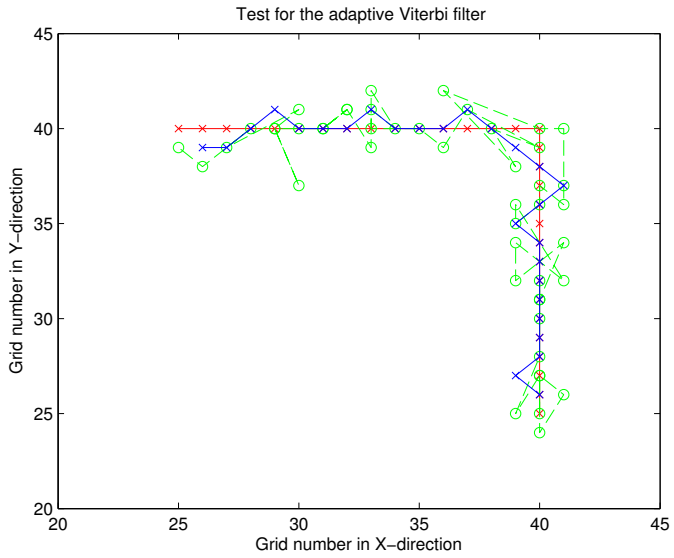

Figure C.9: The results from the adaptive Viterbi filter in the 90° turn test. The blue line is the output from the filter. The red line is the actual path. The green line is the actual path added with noise which is the input for the filter

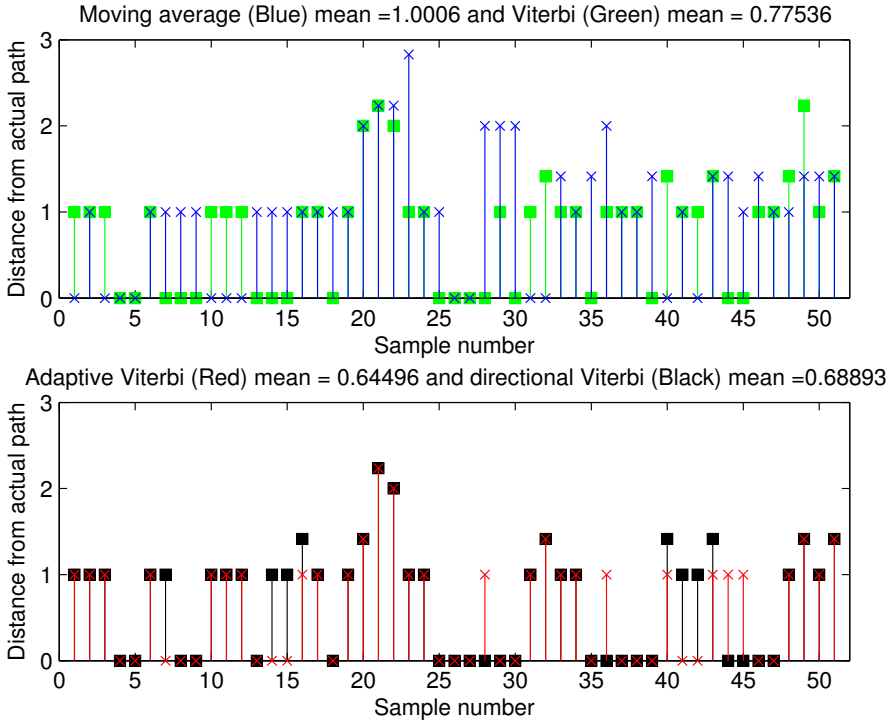

Figure C.10: The error for the four filters in the 90° turn test. The upper graph shows the Moving Average (blue) and the Viterbi (green). The lower graph shows the directional Viterbi (black) and the adaptive Viterbi (red). In both graphs the 90° turn is at sample number 20 - 25.

C.3 180° Turn Test

The 180° turn test is described in Section 4.1 as scenario C. The test is conducted by combining two straight lines from (10,20) to (40,20) and from (40,20) to (10,20).

The adaptive filter is trained by 200 walks added with noise and configured with a forgetting factor of 0.005

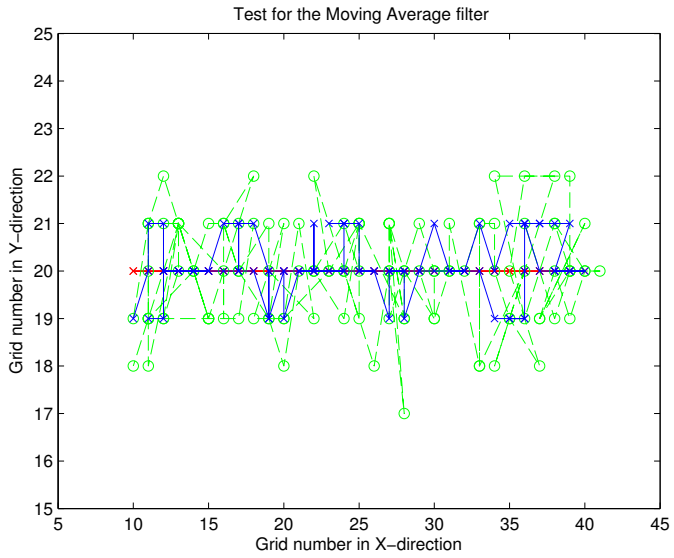

Figure C.11: The results for the Moving Average for the 180° turn test. The blue line is the output from the filter. The green line is actual path added with noise which is the input for the filter. The red line is the actual path

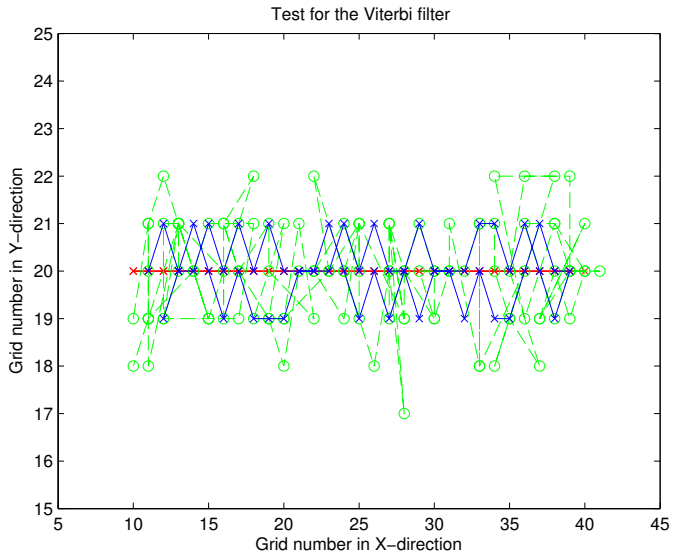

Figure C.12: The results from the Viterbi filter in the 180° turn test. The red line is the actual path. The green is the actual path added with noise and the blue line is the output from the filter

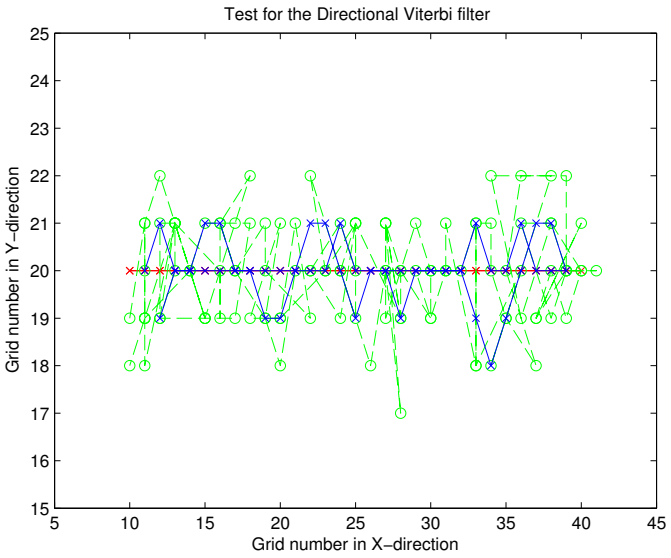

Figure C.13: The directional Viterbi results for the 180° turn test. The blue line is output from the directional Viterbi filter and the green is the input for the filter. The green line is the actual path (red line) added with noise

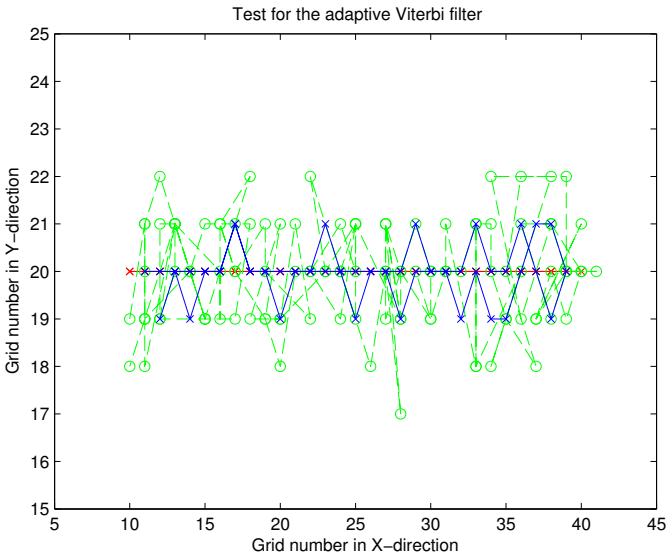

Figure C.14: The adaptive Viterbi filter results for the 180° turn test. The blue line is the output from the filter. The red line is the actual path. The green line is the actual path added with noise which is the input for the filter

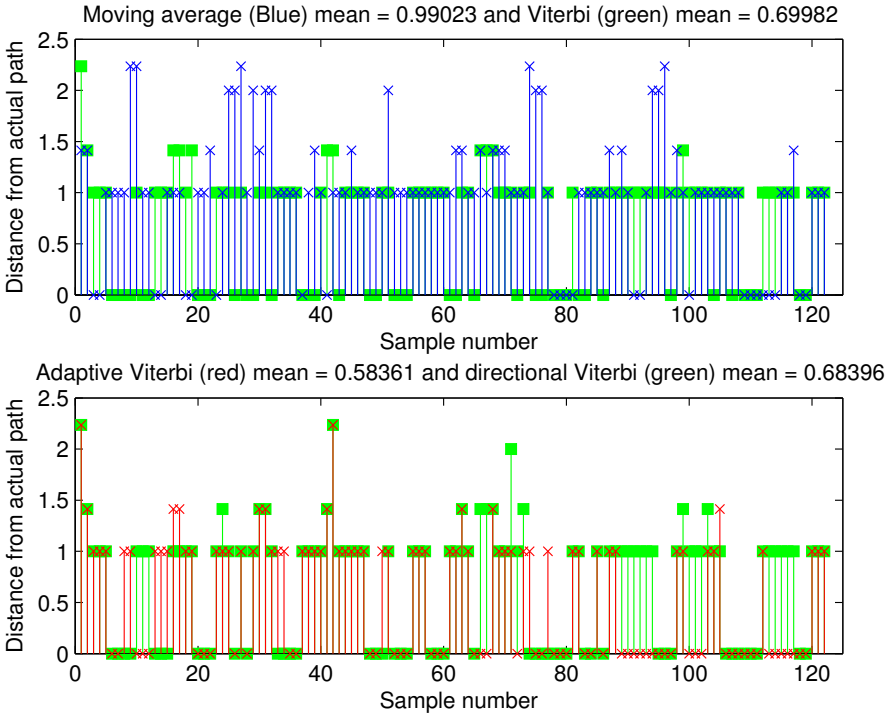

Figure C.15: The error for the four filters in the 180° turn test. The upper graph shows the Moving Average (blue) and the Viterbi (green). The lower graph shows the directional Viterbi (black) and the adaptive Viterbi (red). On both graphs the 180° turn is at sample number 65 - 75.

C.4 Soft Turn Test

The soft turn test is described in Section 4.1 as scenario D. The test is conducted using a circle divided in cells. Like the other tests the adaptive filter is trained using 200 walks and configured with a forgetting factor of 0.005.

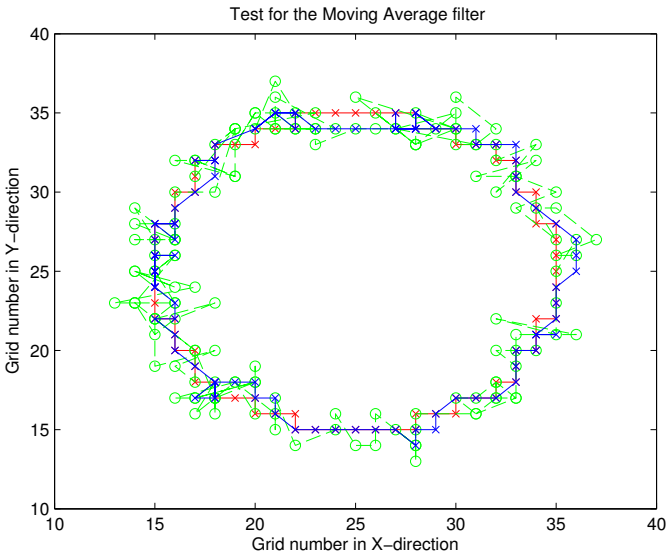

Figure C.16: The Moving Average filter results from the soft turn test. The blue line is the output from the filter. The green line is the input for the filter which is the actual path (red line) added with noise

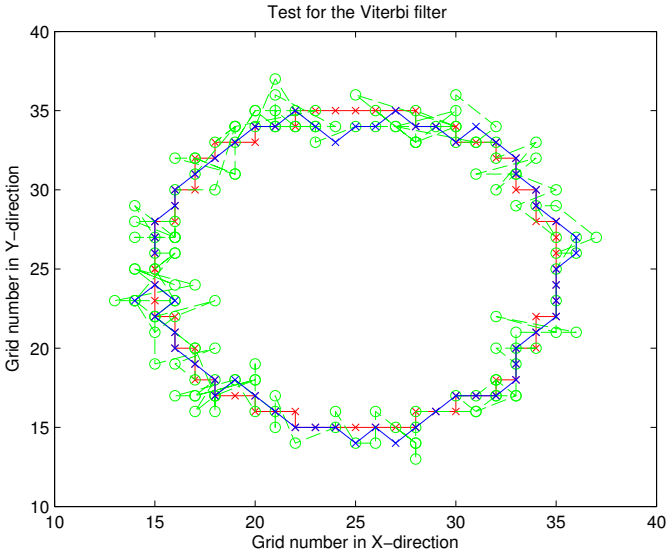

Figure C.17: The Viterbi filter results for the soft turn test. The red line is the actual path and the green is the actual path added with noise which is the input for the filter. The blue line is the output from the filter

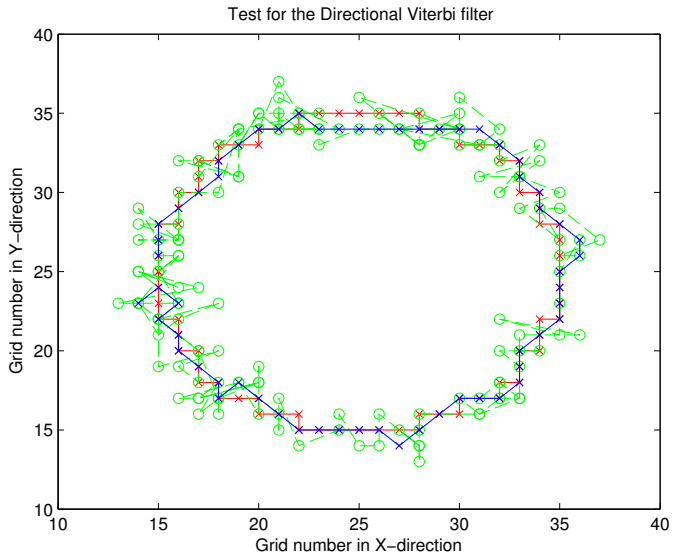

Figure C.18: The directional Viterbi filter results for the soft turn test. The green line is the actual path added with noise. The red line is the actual path. The blue line is the output from the filter.

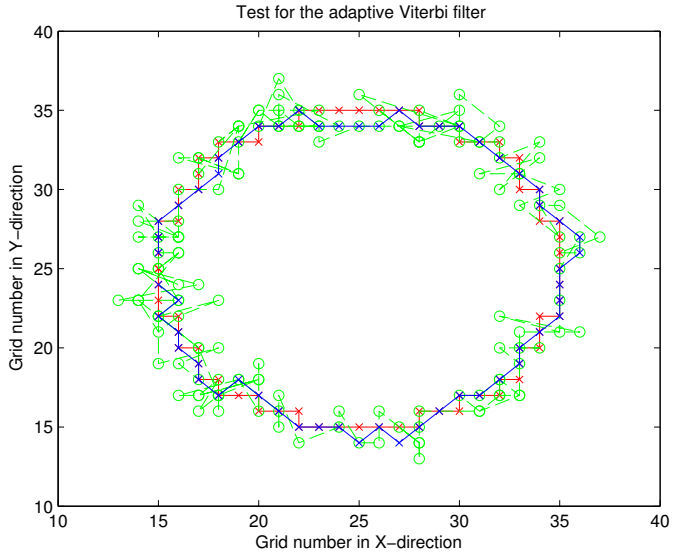

Figure C.19: The adaptive Viterbi filter results for the soft test. The blue line is the output from the filter. The red line is the actual path and the green line is the actual path added with noise which is the input for the filter

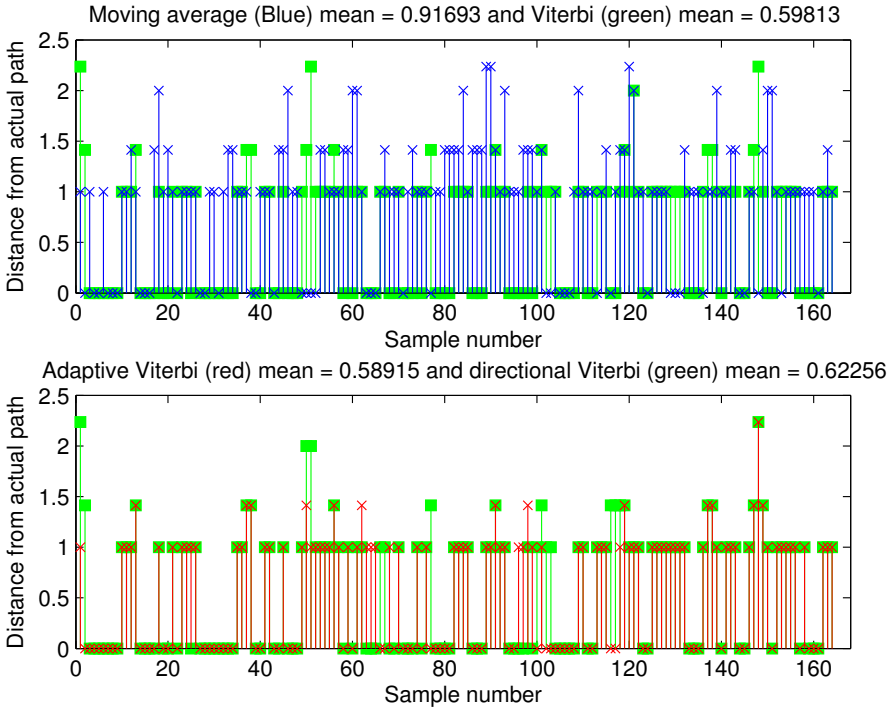

Figure C.20: The error for the filters in the soft turn test. The upper graph shows the Moving Average (blue) and the Viterbi (green). The lower graph shows the directional Viterbi (black) and the adaptive Viterbi (red)

Appendix D

Abbreviations

AOA	Angle Of Arrival
AP	Access Point
CPS	Cambridge Positioning System
GPS	Global Positioning System
GSM	Global System for Mobile Communication
HMM	Hidden Markov Model
LORAN	LOng RAnge Navigation
LOS	Line Of Sight
MD	Mobile Device
PLT	Intel Precision Location Technology
RFID	Radio Frequency IDentification
RSS	Radio Signal Strength
TDoA	Time Difference of Arrival
TOA	Time Of Arrival
Wi-Fi	Wireless Fidelity

Bibliography

[GB00] Walter R. Gilks and Carlo Berzuini. Article - Following a moving target - Monto carlo inference for dynamic Bayesian models. 2000.

[GG05] Fredrik Gustafsson and Fredrik Gunnarsson. Mobile Positioning Using Wireless Networks - Possibilities and fundamental limitations based on available wireless network measurements. 2005.

[Hoo11] Andreas Van Hooijdonk. Many positioning systems exist, 2011. `http://www.gps-practice-and-fun.com/positioning-systems.html`.

[JM00] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[Joh] Richard Johnsonbaugh. *Discrete Mathematics*, chapter 9. Jk Computer Science and Mathematics.

[Les05] Patrick Lester. A* pathfindig for beginnners, 2005. `http://www.policyalmanac.org/games/aStarTutorial.htm`.

[Med10] MediaCart. MediaCart website, 2010. `http://www.mediacart.com`.

[Mil97] Adam Milstein. *Report - Occupancy Grid Maps for Localization and Mapping*. University of Waterloo, 1997.

[Neg11] Christine Negroni. Tracking Your Wi-Fi Trail, 2011. `http://www.nytimes.com/2011/03/22/business/22airport.html?_r=2&ref=technology`.

[NHM+11] Jimmy Jessen Nielsen, Jose Holgado, Tatiana Kozlova Madsen, Claus Pedersen, Bernard Henri Fleury, Francisco Quiros, Igor Arambasic, Na Yi, Yi Ma, Dirk Slock, Benoît Denis, Hadi Noureddine, Loïc Brunel, Damien Castelain, Bernard Uguen, Fernando Rivas, Esther López, Francisco de Arriba, Marios Raspopoulos, George Agapiou, Joaquim Bastos, Senka Hadzic, Hugo Marques, Armin Dammann, Christian Mensing, and Ronald Raulefs. D1.1 scenarios and parameters, 2011.

[PAK+05] Neal Patwari, Joshua N. Ash, Spyros Kyperountas, Alfred O. Hero III, Randolph L. Moses, and Neiyer S. Correal. Locating the Nodes - Cooperative localization in wireless sensor network. 2005.

[Ros04] Sheldon M. Ross. *Introduction to Probability and Statistics for Engineers and Scientists 3. ed.*, pages 293–321. Academic Press, 2004.

[RPN+08] Janne Dahl Rasmussen, Achuthan Paramanathan, Yassine Nassili, Anders Grauballe, and Mikkel Gade Jensen. *Report - Indoor positioning - Based on Bluetooth*. Aalborg University, 2008.

[SCGL05] Guolin Sun, Jie Chen, Wei Guo, and K.J. Ray Liu. Signal Processing Techniques in Network Aided Positioning - A survey of state-of-the-art positioning designs. 2005.

[STK05] Ali H. Sayed, Alireza Tarighat, and Nima Khajehnouri. Network-Based Wireless Loaction - Challenges faced in developing techniques for accurate wireless location information. 2005.

[Thr00] Sebastian Thrun. Article - Probabilistic Algorithm in Robotics. 2000.

[Wik11] Wikipedia. Global positioning system, 2011. `http://en.wikipedia.org/wiki/Global_Positioning_System`.