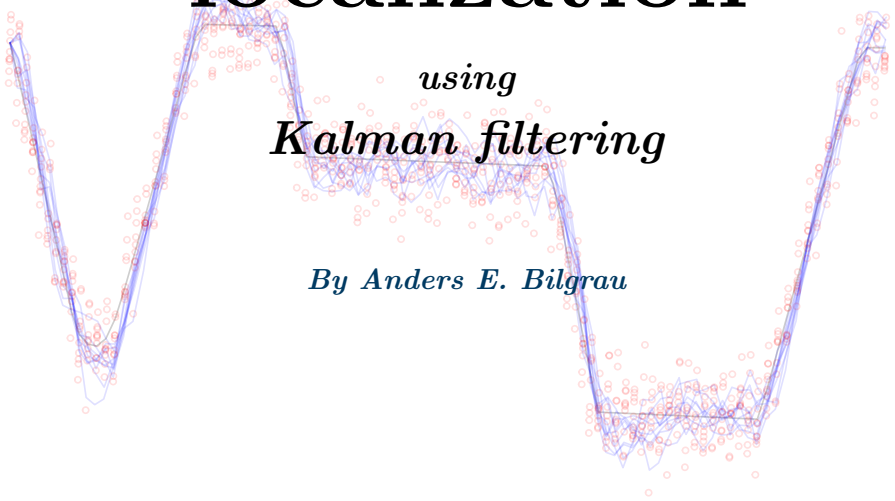
Mobile indoor localization

using

Kalman filtering



By Anders E. Bilgrau

27th of May, 2011

Department of Mathematical Sciences

Supervisors: Jakob G. Rasmussen & Rasmus L. Olsen

In collaboration with Department of Electronics & Information Technology

AALBORG UNIVERSITET

Institut for Matematiske Fag, I-17
Fredrik Bajers Vej 7 G
9220 Aalborg Øst
Telefon: 99 40 88 04
Fax: 98 15 81 29
http://www.math.aau.dk/

Title:
Mobile indoor localization using Kalman Filtering

Topics:
Applied mathematics, Mobile Localization, Kalman Filtering, Indoor Localization, Path finding, Mobility models, Random way-point, Heat equation, Diffusion Equation

Project period:
COMB2, Spring 2011

Author:

Anders Ellern Bilgrau

Supervisors:
Jakob G. Rasmussen
Rasmus L. Olsen

Printed copies: 5
Pagenumber: 56
Appendices: 3
Finished: 27th of May, 2011

Abstract:

This report studies mobile indoor localization using Kalman filters to estimate the current state in a hidden Markov chain. The optimal Kalman filters has been a very successful tool in a wide variety of applications and has also proved useful in localization. The report introduces and then derives the Kalman filter in the context of the larger class of Bayesian filters.
Location data from a faux office is simulated from a variation of the random way-point mobility model using the Heat equation for path finding.
An heuristic attempt to improve the Kalman filter by reusing old inferred positions is done with ambiguous results. The Kalman filter can be slightly improved although it is not worth the extra computational effort.

Preface

This project was prepared in the fall of 2010 at Aalborg University by Anders E. Bilgrau and supervised by Jakob G. Rasmussen, department of Mathematical Sciences, and Rasmus L. Olsen, Department of Electronics & Information Technology.

The report assumes intermediate knowledge and insight in set and probability theory. The reader should also be well-acquainted with standard topics such as functional analysis. No knowledge of measure theory and partial differential equations is required. The report do assume some familiarity with both bi- and univariate kernel density estimation; appendix A is a very brief introduction on this topic.

The primary objective of the report is to study and discuss Kalman filtering as means to infer the current position of some indoor mobile device.

Since we avoid measure theory along with other complex mathematical subjects, the strictly necessary foundations are ignored and thus many mathematical details are not accounted for. The theory and style of the report is therefore heuristic and informal and the report is a compromise between theory and application.

Citations are listed after an IEEE-like style and the sources are seen in the bibliography. All programming and implementation of Kalman filters along with the majority of the all figures were written in the programming language R. [1]

The digital version of the report is best viewed in Adobe Reader 9 or newer. Also, note that the table of content, links, and references are clickable internal hyperlinks. All figure are in vector graphics which is great for zooming in.

Table of Contents

Chapter 1

Introduction

Exact and reliable localization of mobile devices is in high demand and numerous techniques for providing such estimates have been proposed. The method of localizing some electric device of course depends on the technology used. For instance, the Global Positioning System (GPS) computes distances to the satellites using transmission times and derive, via trilateration, an estimate of the position of the device. Bluetooth technology based systems may use signal strengths to arrive at a location estimate. In this report, we discuss indoor location estimation in the context of mobile wireless technologies, e.g. WLAN or Bluetooth, though the results are not necessarily restricted to it.

Throughout the report, the general problem is to infer the *current* position of some mobile device based on noisy and inexact measurements of the current and earlier positions. Obviously, the naïve way of doing this is to use the processed "raw" measurements of the location. However, much of the noise can be removed and even better estimates can be achieved. To elaborate on the mathematical notation of this, let $(x_2, y_2), (x_2, y_2), \ldots$ be the true sequence of positions of the device at time $t_1, t_2, \ldots$. The measurements of the positions (x_i, y_i) are usually derived from other physical quantities which makes these observations inexact. I.e. we can only observe $\boldsymbol{z}_i := (x_i + u_i, y_i + v_i)$ for all i, where u_i and v_i are two random variables. Precisely how u_i and v_i are distributed will be discussed later. To reiterate, the problem estimate the current true position (x_n, y_n) having only the corrupted "raw" observations $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ available. The naïve approach above (i.e. $(\hat{x}_n, \hat{y}_n) := \boldsymbol{z}_n$) obviously ignores a lot of information. For instance, we should expect the successive observations to be correlated spatially and so $\boldsymbol{z}_{n-1}$ is also informative about (x_n, y_n).

We shall concern ourselves with localization of devices handled by humans, such as Smart-phones or laptops, and thus we are effectively estimating the position and movement of humans. This is reflected in the construction of the following presented mobility model.

Very high raw accuracy can be achieved in localizing *static* devices; however, accuracy often results in significantly increased computation times. Such delay is usually unimportant when the device is stationary, while this is certainly not the case when the device is mobile. Obviously, long delays results unreliable location estimates as it is accurate of a *previous* position. In this manner exact and reliable mobile localization becomes a trade-off between delay and precision. While this delay-precision aspect is important, this report does not deal with it in greater detail.

With assumptions on how the device moves noise can be eliminated and better estimates can be achieved. The modelling of the spatial behaviour of the device is called the mobility model. As no real data has been available, we first wish to simulate a data set from a faux office.

1.1 Mobility models

The mobility model, as said, models the behaviour of whatever one is trying to localize. The right choice (if such a thing exists) of mobility model is problem-dependant but since this report is concerned with indoor localization of hand-held devices the mobility model presented in the following should essentially mimic human behaviour. While we shall keep it as simple as possible, mobility models can be made arbitrarily complex taking ever more variables into account. The goal of this section is to construct a realistic human mobility model.

We shall see, that models which takes e.g. environmental obstacles, momentum and speed of movement into account can be somewhat easily made. [2] [3]

The popular so-called *random walk mobility models* is a large class of mobility models which all largely fits into the following simplified algorithm;

i. Choose a direction from the interval $[0, 2\pi]$ uniformly.

ii. Choose a speed from the interval $[v_{min}, v_{max}]$ also uniformly.

iii. Move with the selected speed in the selected direction either

- for some specified distance or
- in some predefined time.

iv. Wait some uniformly distributed time. Then repeat the procedure.

If the area in which the device moves is finite some rules for when the device hits the area boundaries can be imposed on the model. E.g. a new direction can be chosen if the border is hit according to some bouncing rule. The *random direction* mobility model is largely as above but moves in the selected direction at the chosen speed until it hits the border.

If $v_{min} = 0$ the phenomenon speed decay occurs where the average speed converge to zero as time goes to infinity. If $v_{min} > 0$ the average speed converge to some speed above v_{min}. [4]

The *Random Waypoint* model is also a popular mobility model. The random waypoint requires we have a finite convex area for, as we shall see, obvious reasons. The simplest form of such an algorithm goes as follows;

i. Choose a uniformly selected *waypoint* from the set of possible waypoints.

ii. Select a speed v uniformly from some interval $[v_{min}, v_{max}]$.

iii. Move with speed v toward the selected *waypoint* until it is reached.

iv. Wait some uniformly distributed time and then repeat process.

This gives the essential gist of the random waypoint model, which however, can the deviate quite alot from the above procedure. The requirement of finite space in the above is obvious since a waypoint cannot be chosen uniformly from an unbounded set. The convexity ensures the device does not leave the area on its way to the waypoint. The random waypoint model can somewhat easily be expanded to be boundary-less by wrapping the area onto a torus. Note, that more generally the set of possible waypoints may be a finite set of non-random waypoints. Also, the drawing of a waypoint from this set need not necessarily be uniform.

A good, quick, and informal page describing these mobility models can be found at [5].

The speed during the movement may also be much more complex than simply a constant uniformly sampled speed. These so-called *speed models* can e.g. vary the speed over time (with some even stopping), be stochastic or deterministic (or a combination).

Likewise, the *waiting time* (sometimes called *dwell time*) may also be modelled by non-uniform distributions and depend on various parameters. The waiting times are sometimes modelled into the speed model with speeds of zero.

The simple models above obviously does not model human behaviour very well in general. Most notably, they do not take walls and obstacles into account; a huge factor in indoor localization. The discussion therefore turns to path-finding.

1.2 Path finding

As an introduction to path finding, we first consider the simple Lee algorithm which primarily has been used for routing of circuits in printed circuit boards. [6]

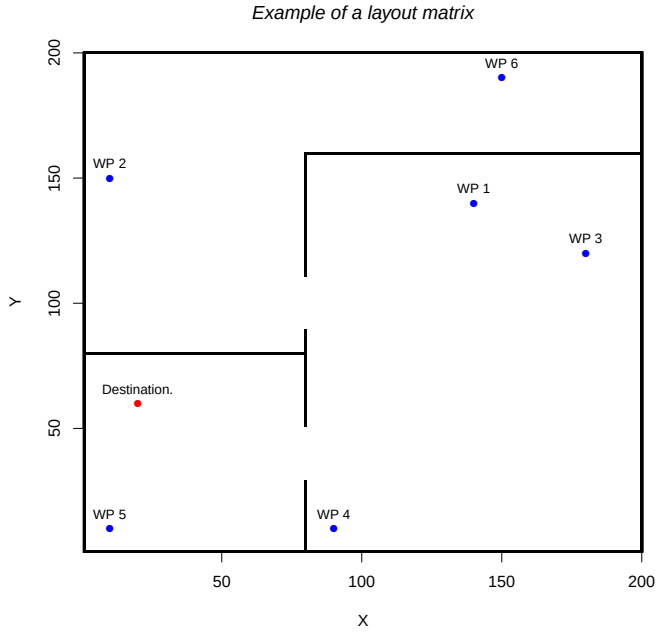


Figure 1.1: *A illustration of a 200 by 200 layout map matrix describing some rooms together with a destination and 6 starting way-points. The inaccessible black squares (in this case walls) corresponds to entry values of zero and white accessible areas to one. In appendix C, the R-script C.1 produces this plot.*

1.2.1 The Lee algorithm

Suppose we are confined to a rectangular area on which we wish to find the path from a starting point (x_s, y_s) and some destination (x_d, y_d). The area is discretized into n by m grid and the accessible regions is characterized by a so-called *layout map matrix* $\boldsymbol{L}$ of size $n \times m$ where each entry $l_{i,j}$ in $\boldsymbol{L}$ corresponds the spatial grid and is defined by

$$l_{i,j} = \begin{cases} 1 & \text{if } l_{i,j} \text{ is accessible} \\ 0 & \text{if } l_{i,j} \text{ is inaccessible} \end{cases} \tag{1.1}$$

Figure 1.1 is an example of a layout map matrix that will be used throughout the report.

The Lee algorithm is now simple to implement. Let $\boldsymbol{A}$ be a $n \times m$ matrix and set value at the *destination point* to one, i.e. $a_{x_d,y_d} := 1$. From the destination a_{x_d,y_d} all "empty", accessible, and direct neighbouring entries (over, under, left, right) are incremented by 1 until the starting point (x_s, y_s) (or all accessible entries) has been given a number. The route is now found by backtracking from the starting point by following the lowest neighbouring points. This procedure illustrated in figure 1.2.

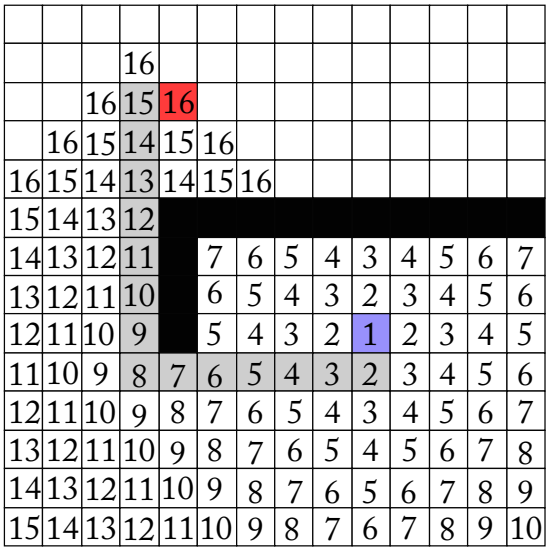


Figure 1.2: *The entries above, below, right, and left to the destination (in blue) are incremented by one until the starting point (red) is reached. The black squares correspond inaccessible places, i.e. where the layout matrix entries* $l_{i,j} = 0$. *The generated path is shown in grey. Note, that this path is not unique.*

Note, that if the incrementation is continued until all accessible entries are filled, the starting point can be picked arbitrarily and the route to the destination found effortlessly in accordance to the above. The point here is that the computational effort lies in computing the gradient field. One way to implement the algorithm is seen in R-script C.1.

Figure 1.3 uses the Lee algorithm on the layout map of figure 1.1. From figure 1.3 it is seen that the paths creep along walls which intuitively not model human mobility very well. Also, the paths are very rough and too linear as human paths. To overcome this problem we adopt a similar general concept of following the gradient albeit with a much more complicated process to generate the gradient field; namely the heat (or diffusion) equation.

1.2.2 Using the heat equation

The heat equation is a partial differential equation which describes the temperature of a system at some point in space and time. The idea here is to let the destination be a heat source from which heat flows out and fills the room. With the rooms heated, one can follow the line of highest rising temperatures to the source.

The equation is also known as both *gas* and *diffusion equation* since it also describes both the micro- and macroscopic diffusion of particles in fluids (or gas) as the probability density and concentration respectively. Here, however, we keep the using the equation in the context of heat.

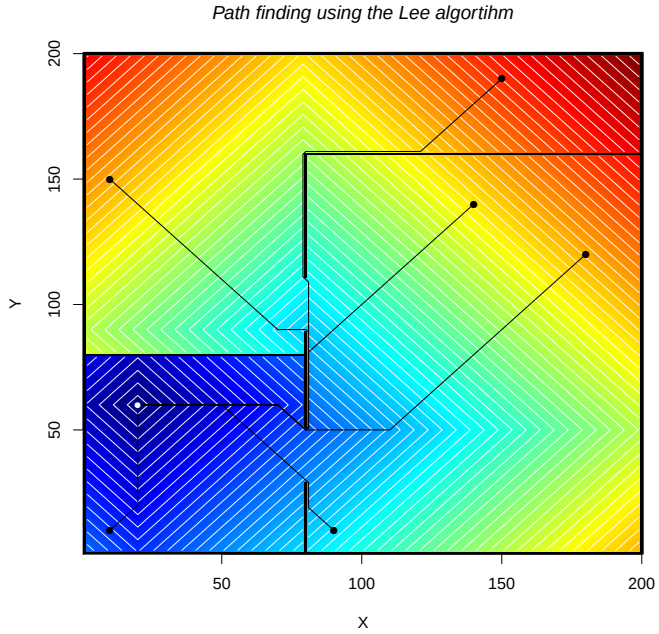


Figure 1.3: *The paths from the waypoints (black) to the destination (white) generated by the Lee algorithm using the layout matrix of figure 1.1. The colouring illustrates the incrementation from the destination. Contour lines are shown in white to which the path always runs perpendicular. This plot is outputted from R-script C.1.*

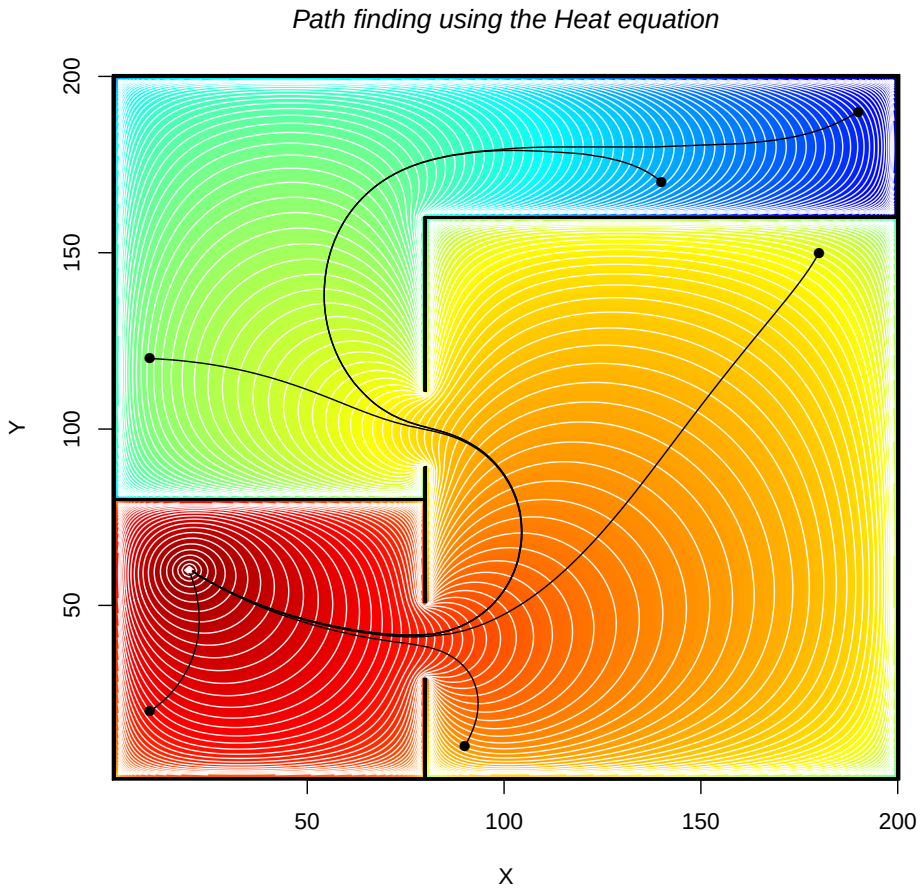


Figure 1.4: *The generated paths from the waypoints (black) to the destination (white) by using the heat equation and the layout map from figure 1.1. The colouring illustrates the falling temperatures that radiates from the destination. As above, the path always runs perpendicular to the contour lines shown in white. This plot is outputted from R-script C.2*

If $u(x, y, t)$ is the temperature at the spatial location (x, y) at time t then is satisfies the heat equation given by

$$\frac{\partial u}{\partial t} - \kappa \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0,$$

where κ is some constant scalar. The solution $u(x, y, t)$ is approximated in by discrete points (x_i, y_j, t_k) in accordance to the above. The rectangular region $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is split into an Δs-equidistant mesh $\{x_1, x_2, \ldots, x_n\} \times \{y_1, y_2, \ldots, y_m\}$. I.e. we have $x_{i+1} = x_i + \Delta s$ and $y_{j+1} = y_j + \Delta s$ for all i and j. The temporal dimension is also discretized by letting $t_k = k \cdot \Delta t$ for all $k = 0, 1, 2, \ldots$ and for some Δt.

If $u_{i,j}^k$ is the temperature at (x_i, y_j) at time t_k then partial derivatives above can be approximated by

$$\begin{aligned} \frac{\partial u}{\partial t} &\approx \frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t}, \\ \frac{\partial^2 u}{\partial x^2} &\approx \frac{u_{i+1,j}^k + u_{i-1,j}^k - 2u_{i,j}^k}{\Delta s^2}, \quad \text{and} \\ \frac{\partial^2 u}{\partial y^2} &\approx \frac{u_{i,j+1}^k + u_{i,j-1}^k - 2u_{i,j}^k}{\Delta s^2}. \end{aligned}$$

These approximations are derived by applying the so-called forward and central finite difference approximations[1]. Substituting the approximations into the heat equation we get

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} = \kappa \left(\frac{u_{i+1,j}^k + u_{i-1,j}^k - 2u_{i,j}^k}{\Delta s^2} + \frac{u_{i,j+1}^k + u_{i,j-1}^k - 2u_{i,j}^k}{\Delta s^2} \right)$$

which implies

$$u_{i,j}^{k+1} = u_{i,j}^k + \kappa \frac{\Delta t}{\Delta s^2} \left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - 4u_{i,j}^k \right) \tag{1.2}$$

It is seen that at a given point the rate of change in temperature is directly proportional to the difference between the average temperature around the point and the temperature at the point. This becomes apparent by rearranging the above

$$\frac{u_{i,j}^{k+1} - u_{i,j}^k}{\Delta t} = \kappa' \left(\frac{u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k}{4} - u_{i,j}^k \right),$$

[1] If $f(x)$ is a differentiable function, then the derivative can be approximated by

$$\begin{aligned} f'(x) &\approx \frac{f(x+h) - f(x)}{h} \\ f'(x) &\approx \frac{f\left(x + \frac{1}{2}h\right) - f\left(x - \frac{1}{2}h\right)}{h}, \end{aligned}$$

called the forward and central differences. The second order partial derivatives above is obtained by applying the central difference twice.

where $\kappa' = \frac{4}{\Delta s^2}\kappa$. [7]

The procedure for the heat equation is now nearly identical to the Lee algorithm. However, instead of the simple incrementation the value of nearby (up, down, left, right) points are computed by using equation (1.2). The algorithm in pseudo-code is as follows. Let $\boldsymbol{u}^0$ be a $n \times m$ matrix describing the temperature at time $k = 0$ with $u^0_{x_d,y_d} = 1$ and all other entries zero. We then approximate the solution $u(x, y, t)$ at different times by computing the heat matrices $\boldsymbol{u}^1, \boldsymbol{u}^2, \ldots$ for the next time instances $k = 1, 2, \ldots$;

for $k = 0, 1, 2, 3, \ldots$ do

Set $u^k_{x_d,y_d} := 1$

for $i = 2, 3 \ldots, n-1$ do

for $j = 2, 3 \ldots, m-1$ do

According to equation (1.2), let

$$u^{k+1}_{i,j} := u^k_{i,j} + \kappa \frac{\Delta t}{\Delta s^2} \left(u^k_{i+1,j} + u^k_{i-1,j} + u^k_{i,j+1} + u^k_{i,j-1} - 4u^k_{i,j} \right).$$

end do

end do

Let $\boldsymbol{u}^{k+1} := \boldsymbol{L} \circ \boldsymbol{u}^k$, where $\circ$ denotes the Hadamard product (or element-wise matrix product) and $\boldsymbol{L}$ is the layout map matrix.

end do

By constantly keeping the temperature at the source to 1 and multiplying element-wise with the layout matrix $\boldsymbol{L}$ ensures that the source and wall is radiating and absorbing heat respectively. [8] An implementation in the language of R is seen in script C.2.

As can be seen in figure 1.4, the paths generated by using the heat equation seems much more plausible as human paths. It should be noted, that path e.g. from way-point one to two may not coincide with that two to one. Figure 1.5 shows all possible paths between 11 way points.

1.3 The used mobility & observation model

In this report, the used mobility model is the above random waypoint model combined with path finding using the heat equation and the following speed and waiting models. Assuming we have a collection of n_w waypoints,

$$\mathcal{W} = \left\{ w_1{=}(x_1, y_1),\ w_2{=}(x_2, y_2),\ \ldots,\ w_{n_w}{=}(x_{n_w}, y_{n_w}) \right\},$$

from which random way-points are sampled $w_{p_1}, w_{p_2}, w_{p_3}, \ldots$ such that $w_{p_i} \neq w_{p_{i+1}}$. I.e. the same way-point is not chosen twice in a row. The random

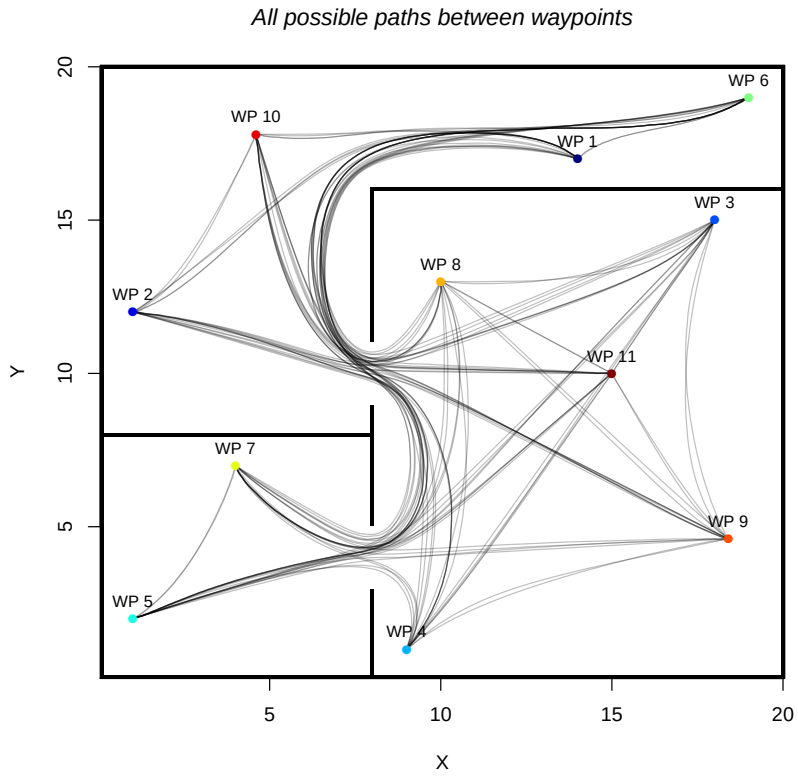


Figure 1.5: *All 110 possible paths between the 11 way-points.*

path is now given by the above sequence connected by the paths given by the heat equation as described above. The way-points and path used for the simulation is the 11 way-points and 110 paths of figure 1.5.

When each way-point is reached, a waiting time t_i at way-point w_i is simulated by exponential distributed; i.e. $t_i \sim \text{Exp}(\lambda_i)$. For the data simulation, we use $\lambda_1 = \lambda_2, \ldots = \frac{1}{20}$; thus on average we wait 20 seconds at each way-point. As I could find no experimental data on how actual waiting times are distributed this is an arbitrary choice that sounds somewhat fair and is mathematical convenient. In reality, heavier tailed (perhaps even multi-modal) distributions is probably the case.

The average walking speed for humans is about $5km/h \approx 1.38m/s$. [9] As we do not allow for negative speed, the speed model is simply drawn from a (truncated) Gaussian distribution (with support $[0, \infty)$) with mean 5 and standard deviation 0.25.

For the observed quantities we simply corrupt the true path, sampled with at rate of 1 measurement per second, by additive zero-mean Gaussian white noise with standard deviation $\sigma = \frac{1}{2}$; i.e. about 95% of the observed locations are within 1 metre of the true position. In reality, this depends on the number of access points (and thus delay) as mentioned in the beginning of this chapter and seen in [10].

Figure 1.6 shows the simulated data produced by R-script C.5 according to the above. The randomly chosen sequence of way-points was

$$\begin{aligned}\big(&3, 6, 1, 10, 6, 10, 3, 5, 7, 4, 5, 2, 1, 9, 7, 4,\\ &2, 8, 11, 3, 4, 3, 10, 1, 2, 6, 8, 6, 8, 6, 4, 6,\\ &9, 7, 6, 3, 5, 6, 10, 9, 8, 10, 4, 10, 7, 11, 1\big)\end{aligned}$$

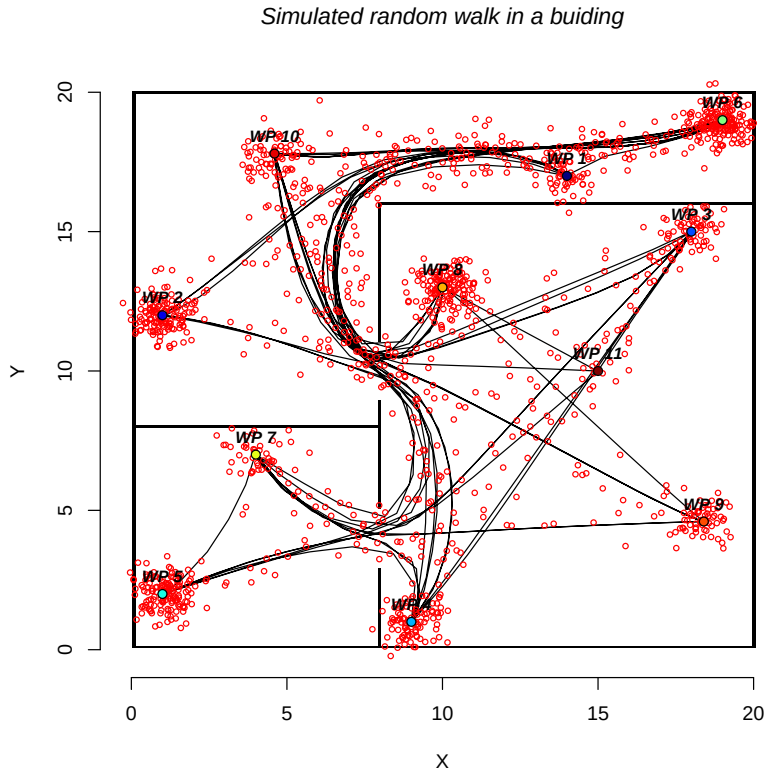


Figure 1.6: *The simulated data of the random walk between 11 way-points as described above. Black solid lines show the true path and the red circles are the measurements.*

Chapter 2

Kalman filtering

Kalman filters, due to Rudolf E. Kalman's famous paper [11], have since its publication in 1960 become a hugely popular filter used in a wide spectrum of applications. They are *filters* in the sense that they filter out noise.

Denote by $\boldsymbol{x}_k \in \mathbb{R}^n$ the *state* vector of some dynamical system. Ultimately, we are interested in estimating this state vector $\boldsymbol{x}_k$ which characterize the current state of the system. It is assumed, that the system is governed by a difference equation of the form

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{w}_k, \tag{2.1}$$

where the *state transition matrix* $\boldsymbol{F}_k$ denotes a $n \times n$ matrix, the $n \times n$ matrix $\boldsymbol{B}_k$ describes the known *control-input* given by the *control vector* $\boldsymbol{u}_k \in \mathbb{R}^n$, and $\boldsymbol{w}_k \in \mathbb{R}^n$ is *process noise* assumed to multivariate normal, i.e. $\boldsymbol{w}_k \sim N_n(\boldsymbol{0}, \boldsymbol{Q}_k)$. Equation (2.1) is sometimes also referred to as the *state space model*. The state vector, in our specific case, will contain the position coordinates among other quantities.

The state vector is, however, not directly observable and can only be inferred though the use of the so-called *observation model* that maps the state space into the *observation space*;

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k, \tag{2.2}$$

where $\boldsymbol{H}_k$ is the *observation model*, $\boldsymbol{v}_k$ is *measurement noise* assumed to be $N_n(\boldsymbol{0}, \boldsymbol{R}_k)$-distributed.

The initial states, measurement noise and process noise, i.e. the set $\{\boldsymbol{x}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_k, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_k\}$, are assumed mutually independent.

The estimate of the current state vector $\boldsymbol{x}_i$ can obviously only be computed from the available measurements $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_i$. We denote by $\hat{\boldsymbol{x}}_{i|j}$ the estimate of $\boldsymbol{x}_i$ given all observation up to time j i.e. the set $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_j\}$. The Kalman filter operates recursively in two conceptualized phases called the *prediction* and *update step* in which the filter predicts the *a priori state*

estimate, denoted by $\hat{\boldsymbol{x}}_{k|k-1}$, and then updates the *a posteriori state estimate* $\hat{\boldsymbol{x}}_{k|k}$. Here a priori and a posteriori refers to the observation of $\boldsymbol{z}_k$.

To access the estimated accuracy of the state estimate both *a priori* and *a posteriori error covariance matrices*, denoted by $\boldsymbol{P}_{k|k-1}$ and $\boldsymbol{P}_{k|k}$ respectively, is also computed.

2.1 Prediction step

Since the filter works recursively, assume that $\hat{\boldsymbol{x}}_{k-1|k-1}$ and $\boldsymbol{P}_{k-1|k-1}$ are available; then the a priori state estimate is computed by the equation

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}_k \hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{B}_k \boldsymbol{u}_k,$$

along with the a priori error covariance matrix

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_k. \tag{2.3}$$

Again, the a priori error covariance matrix $\boldsymbol{P}_{k|k-1}$ describes the accuracy of $\hat{\boldsymbol{x}}_{k|k-1}$. These a priori estimates are used in the updating step of the Kalman filter.

2.2 Update step

For simplicity of the updating step, some helpful quantities are computed first;

$$\begin{aligned}
\tilde{\boldsymbol{y}}_k &= \boldsymbol{z}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}}_{k|k-1} && \text{(Measurement residual)} \\
\boldsymbol{S}_k &= \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k && \text{(Residual covariance)} \\
\boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1} && \text{(Optimal Kalman Gain).}
\end{aligned}$$

From these the a posteriori state estimate is given by

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k, \tag{2.4}$$

and the a posteriori error covariance estimate,

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1}, \tag{2.5}$$

can be computed. [12] [13] It is shown later in what sense the Kalman filter is optimal. First, we shall see what the different matrices of the observation and state space model look like in the context of localization.

2.3 An implementation of a Kalman filter

The specific Kalman filter is presented here can be used to infer the present position of some device on the basis of earlier position measurements; which is precisely what we are interested in! From the laws of motion in classical mechanics we can construct a state model. The relationship between position s as a function of time and it's derivatives; speed $v = \frac{ds}{dt}$, acceleration $a = \frac{dv}{dt}$, and jerk $j = \frac{da}{dt}$. The three formulas we need here, is

$$\frac{d^3s}{dt^3} = j, \qquad \frac{d^2v}{dt^2} = j, \qquad \text{and} \qquad \frac{dv}{da} = j. \tag{2.6}$$

Note that jerk is nothing more than the rate of change in acceleration. By taking anti-derivatives on both sides of the first formula we can informally derive a useful expression;

$$\begin{aligned} s &= \iiint j\, dtdtdt = \iint \left(jt + a_0\right) dtdt \\ &= \int \left(\frac{1}{2}jt^2 + a_0t + v_0\right) dt = \frac{1}{6}jt^3 + \frac{1}{2}a_0t^2 + v_0t + s_0, \end{aligned}$$

where s_0, v_0, a_0 are the start position, speed, and acceleration. Similarly the last two equations of (2.6) yield

$$\begin{aligned} v &= \frac{1}{2}jt^2a_0t + v_0, \\ a &= jt + a_0. \end{aligned}$$

Now, by substituting t with Δt_k in the above and letting $j_k \sim N(0, \sigma_k^2)$ we can summarize the above equations of motion with the system

$$\boldsymbol{x}_k = \boldsymbol{F}_k\boldsymbol{x}_{k-1} + \boldsymbol{G}_k j_k$$

where

$$\boldsymbol{F}_k = \begin{bmatrix} 1 & \Delta t_k & \frac{1}{2}\Delta t_k^2 & 0 & 0 & 0 \\ 0 & 1 & \Delta t_k & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t_k & \frac{1}{2}\Delta t_k^2 \\ 0 & 0 & 0 & 0 & 1 & \Delta t_k \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{x}_k = \begin{bmatrix} x_k \\ x_k' \\ x_k'' \\ y_k \\ y_k' \\ y_k'' \end{bmatrix}, \quad \boldsymbol{G}_k = \begin{bmatrix} \frac{1}{6}\Delta t_k^3 \\ \frac{1}{2}\Delta t_k^2 \\ \Delta t_k \\ \frac{1}{6}\Delta t_k^3 \\ \frac{1}{2}\Delta t_k^2 \\ \Delta t_k \end{bmatrix},$$

and $\Delta t_k = t_k - t_{k-1}$. It is thus assumed that in the timespan of Δt_k the jerk j_k is constant and equal in both directions. This is not yet a state model, as components of the term $\boldsymbol{G}_k j_k$ is completely determined by any of the others. I.e. $\boldsymbol{G}_k j_k$ is just a stochastically *scaled* vector and not a stochastic vector.

With inspiration from $\mathbb{E}[j_k \boldsymbol{G}_k \boldsymbol{G}_k^T]$ let $\boldsymbol{w}_k \sim N_6(\boldsymbol{0}, \boldsymbol{Q}_k)$ be the process noise term where

$$\boldsymbol{Q}_k = \begin{bmatrix} \frac{1}{36}\Delta t_k^6 & \frac{1}{12}\Delta t_k^5 & \frac{1}{6}\Delta t_k^4 & 0 & 0 & 0 \\ \frac{1}{12}\Delta t_k^5 & \frac{1}{4}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 & 0 & 0 & 0 \\ \frac{1}{6}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 & \Delta t_k^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{36}\Delta t_k^6 & \frac{1}{12}\Delta t_k^5 & \frac{1}{6}\Delta t_k^4 \\ 0 & 0 & 0 & \frac{1}{12}\Delta t_k^5 & \frac{1}{4}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 \\ 0 & 0 & 0 & \frac{1}{6}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 & \Delta t_k^2 \end{bmatrix}.$$

This concludes the construction of the state space model

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{w}_k.$$

In this model the state is characterized by the position, speed, and acceleration in both directions.

Knowing that the device moves in two spatial dimensions both an x and y coordinate is obtained for each measurement; i.e. we only observe a corruption $\boldsymbol{z}_k$ of the current position coordinates (the first and fourth component) of $\boldsymbol{x}_k$. This can be expressed as the observation model given by

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k, \qquad \boldsymbol{v}_k \sim N_2(\boldsymbol{0}, \boldsymbol{R})$$

where

$$\boldsymbol{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{R} = \mathbb{E}[\boldsymbol{v}_k \boldsymbol{v}_k^T] = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}.$$

This is but one example of a possible framework; one might stop at the speed and let the acceleration be the driving process noise. Alternatively in other circumstances, one might go even deeper than the jerk and use the (sometimes) so-called jounce. Note that in this model, there is no control term.

2.4 Extended Kalman filter

The need for linearity in the Kalman filter is obviously somewhat restrictive. The extended Kalman filter generalizes the ordinary Kalman filter and allows for non-linear state-space and observation models. It has become the de-facto standard when non-linear estimation state estimation including GPS and other navigation system. [14] Now, if f and h are differentiable functions then

$$\begin{aligned} \boldsymbol{x}_k &= f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \\ \boldsymbol{z}_k &= h(\boldsymbol{x}_k) + \boldsymbol{v}_k, \end{aligned} \tag{2.7}$$

where $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ both are zero-mean multivariate Gaussian with $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ as covariance matrices respectively. Note, in some texts f and h are functions also of the process and measurement noise.

The formulas for the update and prediction steps does not change much. The update step is described by

$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k)$$
$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1|k-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_k$$

along with the prediction step by;

$$\begin{aligned}
\tilde{\boldsymbol{y}}_k &= \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) && \text{(Measurement residual)} \\
\boldsymbol{S}_k &= \boldsymbol{H}_k \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k && \text{(Residual covariance)} \\
\boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1} \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1} && \text{(Optimal Kalman Gain)} \\
\hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k \\
\boldsymbol{P}_{k|k} &= (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k|k-1}.
\end{aligned}$$

Here $\boldsymbol{F}_k$ and $\boldsymbol{H}_k$ are the defined as the Jacobian matrices of f and h,

$$\begin{aligned}
\boldsymbol{F}_k &:= \left. \frac{\partial f}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}=\boldsymbol{u}_{k-1}} \\
\boldsymbol{H}_k &:= \left. \frac{\partial h}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_{k|k-1}}.
\end{aligned} \tag{2.8}$$

From these formulas, it is evident that the extended filter is a linearization of the state transition function and the observation model around the last inferred state. The generalization to the non-linear systems allowed by the extended Kalman filter comes at a price. The extended filter is no longer optimal.

2.5 Derivation of the Kalman filter

The Kalman filter has been derived in various ways. Some, such as [15] and [13], is based on the concept of innovations. Following [16], we shall here derived the equations from the more general method of Recursive Bayesian Estimation.

2.5.1 Recursive Bayesian Estimation

Recursive Bayesian estimation or simply Bayes filtering uses Bayesian theory to recursively estimate an unknown probability density function. The setup assumes that some state process is a first order Markov chain[1] and that

[1] If $X_0, X_0, \ldots$ is a sequence of random variables with values in the state-space Ω and if the Markov property

$$P(X_{n+1} = \boldsymbol{x}_{n+1} | X_n = \boldsymbol{x}_n, \ldots, X_0 = \boldsymbol{x}_0) = P(X_{n+1} = \boldsymbol{x}_{n+1} | X_n = \boldsymbol{x}_n)$$

holds for all $\boldsymbol{x}_n, \ldots, \boldsymbol{x}_0 \in \Omega$ for all n, then X_n is a Markov chain. Paraphrased, the Markov property states that conditioned on the present, the future is independent of the past. [17]

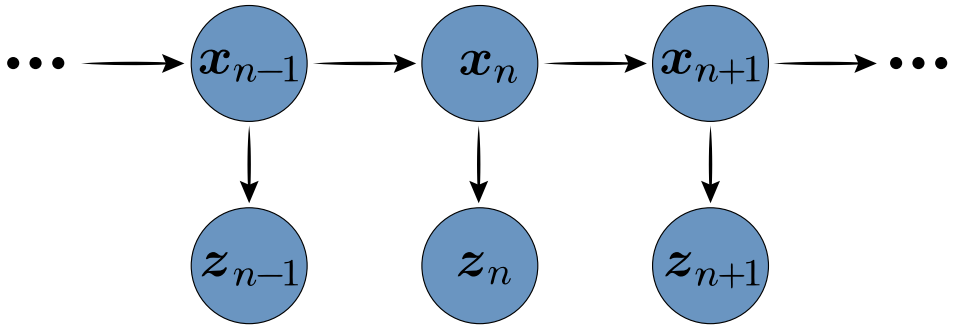


Figure 2.1: *A visualization of the hidden Markov model showing the dependence structure of the variables. The states of the underlying Markov model* $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \ldots\}$ *is not directly observable, as noted elsewhere.*

the k'th observation $\boldsymbol{z}_k$ is independent of all states except $\boldsymbol{x}_k$. From these assumptions, the Markov property

$$p(\boldsymbol{x}_n|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) = p(\boldsymbol{x}_n|\boldsymbol{x}_{n-1})$$

holds for all n and

$$p(\boldsymbol{z}_n|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) = p(\boldsymbol{z}_n|\boldsymbol{x}_{n-1}).$$

This setup, as seen in figure 2.1, forms a so-called hidden Markov model as the states of the Markov chain is unobservable. For simplicity, denote the set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ by $\boldsymbol{x}_{1:k}$ and likewise for $\boldsymbol{z}_{1:k}$.

From Bayes' theorem

$$\begin{aligned}
p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n}) &= \frac{p(\boldsymbol{z}_{1:n}|\boldsymbol{x}_n)p(\boldsymbol{x}_n)}{p(\boldsymbol{z}_{1:n})} \\
&= \frac{p(\boldsymbol{z}_n, \boldsymbol{z}_{1:n-1}|\boldsymbol{x}_n)p(\boldsymbol{x}_n)}{p(\boldsymbol{z}_{1:n-1}, \boldsymbol{z}_n)} \\
&= \frac{p(\boldsymbol{z}_n|\boldsymbol{z}_{1:n-1}, \boldsymbol{x}_n)p(\boldsymbol{z}_{1:n-1}|\boldsymbol{x}_n)p(\boldsymbol{x}_n)}{p(\boldsymbol{z}_n|\boldsymbol{z}_{1:n-1})p(\boldsymbol{z}_{1:n-1})} \\
&= \frac{p(\boldsymbol{z}_n|\boldsymbol{z}_{1:n-1}, \boldsymbol{x}_n)p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1})p(\boldsymbol{z}_{1:n-1})p(\boldsymbol{x}_n)}{p(\boldsymbol{z}_n|\boldsymbol{z}_{1:n-1})p(\boldsymbol{z}_{1:n-1})p(\boldsymbol{x}_n)} \\
&= \frac{p(\boldsymbol{z}_n|\boldsymbol{x}_n)p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1})}{p(\boldsymbol{z}_n|\boldsymbol{z}_{1:n-1})}
\end{aligned} \tag{2.9}$$

where the last step uses the independence of observations along with some terms cancelling. As the usual Bayesian statistics go, this can be interpreted as posterior $= \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$, where the evidence is seen as a normalizing

constant. The prior and evidence can be computed using

$$p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1}) = \int p(\boldsymbol{x}_n|\boldsymbol{x}_{n-1})p(\boldsymbol{x}_{n-1}|\boldsymbol{z}_{1:n-1})d\boldsymbol{x}_{n-1}$$
$$p(\boldsymbol{z}_n|\boldsymbol{z}_{1:n-1}) = \int p(\boldsymbol{z}_n|\boldsymbol{x}_n)p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1})d\boldsymbol{x}_n,$$

respectively; both of which is given by the law of total probability.

The computation of the prior, evidence and posterior is essentially the Bayes filter. In general, though, the calculation of the posterior probability distribution, and thus an optimal Bayesian solution, is only possible in principle as the integrals are unmanageable in practice. Is it feasible only for some restricted special cases; one of which is the Kalman filter.

Out of the various optimal criteria that can be chosen the maximum a posterior (MAP) is the interesting one in the following. As suggested by the name, the MAP estimate is found from the calculated posterior distribution as the point maximizing the distribution. While the MAP is sometimes unreliable, e.g. when the posterior not unimodal (and so the MAP estimate may not even be unique), we need not worry about such problems here as we only deal with the well-behaved Gaussian distributions in the following.

2.5.2 Kalman filter as a special case of the Bayes filter

This derivation of the Kalman filter expands the assumptions of the Bayes filter above and assumes that the process noise $\boldsymbol{w}_n$ and observation noise $\boldsymbol{v}_n$ are zero-mean multivariate Gaussian with covariance matrices

$$\mathbb{E}[\boldsymbol{w}_i\boldsymbol{w}_j^T] = \boldsymbol{Q}_i\delta_{ij} \quad \text{and} \quad \mathbb{E}[\boldsymbol{v}_i\boldsymbol{v}_j^T] = \boldsymbol{R}_i\delta_{ij}, \forall i,j \in \{1,\dots,n\}$$

receptively, where δ_{ij} denotes the Kronecker delta function[2]. Additionally the cross-covariances $\mathbb{E}[\boldsymbol{v}_i\boldsymbol{w}_j^T] = \boldsymbol{0}$. So, it is assumed that all noises are uncorrelated.

Furthermore, we assume that the state and observation processes adhere to the linear relationships of (2.1) and (2.2) respectively for some matrices $\boldsymbol{F}_n$, $\boldsymbol{B}_n$, and $\boldsymbol{H}_n$.

With linearity and the Gaussian assumptions we can find the expected value and covariance of $\boldsymbol{z}_n|\boldsymbol{x}_n$ seen in equation (2.9);

$$\mathbb{E}[\boldsymbol{z}_n|\boldsymbol{x}_n] = \mathbb{E}[\boldsymbol{H}_n\boldsymbol{x}_n + \boldsymbol{v}_n|\boldsymbol{x}_n] = \boldsymbol{H}_n\boldsymbol{x}_n \quad \text{and}$$
$$Cov[\boldsymbol{z}_n|\boldsymbol{x}_n] = Cov[\boldsymbol{v}_n|\boldsymbol{x}_n] = \boldsymbol{R}_n.$$

[2]The Kronecker delta function is defined as

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

All $\boldsymbol{x}_n$ and $\boldsymbol{z}_n$ are Gaussian since they are linear combinations of Gaussian distributed random variables. Thus,

$$p(\boldsymbol{z}_n|\boldsymbol{x}_n) = (2\pi)^{-\frac{N_z}{2}}|\boldsymbol{R}_n|^{-\frac{1}{2}}e^{-\frac{1}{2}(\boldsymbol{z}_n-\boldsymbol{H}_n\boldsymbol{x}_n)^T\boldsymbol{R}_n^{-1}(\boldsymbol{z}_n-\boldsymbol{H}_n\boldsymbol{x}_n)} \tag{2.10}$$

Similarly, we can derive the conditional probability distribution function for $\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1}$ from the expectation

$$\begin{aligned}\mathbb{E}[\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1}] &= \mathbb{E}[\boldsymbol{F}_n\boldsymbol{x}_{n-1} + \boldsymbol{B}_n\boldsymbol{u}_n + \boldsymbol{w}_n|\boldsymbol{z}_{1:n-1}] \\ &= \boldsymbol{F}_n\underbrace{\mathbb{E}[\boldsymbol{x}_{n-1}|\boldsymbol{z}_{1:n-1}]}_{:=\hat{\boldsymbol{x}}_{n-1|n-1}} + \boldsymbol{B}_n\boldsymbol{u}_n := \hat{\boldsymbol{x}}_{n|n-1},\end{aligned}$$

and covariance

$$\begin{aligned}Cov[\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1}] &= \mathbb{E}\big[(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1})(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1})^T\big|\boldsymbol{z}_{1:n-1}\big] \\ &= Cov[\boldsymbol{e}_{n|n-1}] := \boldsymbol{P}_{n|n-1},\end{aligned}$$

where $\boldsymbol{e}_{n|n-1} := \boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1}$. With $\boldsymbol{P}_{n|n-1}$ defined, $\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1}$ follows the Gaussian distribution,

$$p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n-1}) = (2\pi)^{-\frac{N_x}{2}}|\boldsymbol{P}_{n|n-1}|^{-\frac{1}{2}}e^{-\frac{1}{2}(\boldsymbol{x}_n-\hat{\boldsymbol{x}}_{n|n-1})^T\boldsymbol{P}_{n|n-1}^{-1}(\boldsymbol{x}_n-\hat{\boldsymbol{x}}_{n|n-1})} \tag{2.11}$$

Substituting equations (2.10) and (2.11) into (2.9) yields

$$p(\boldsymbol{x}_n|\boldsymbol{z}_{1:n}) \propto e^{-\frac{1}{2}(\boldsymbol{x}_n-\hat{\boldsymbol{x}}_{n|n-1})^T\boldsymbol{P}_{n|n-1}^{-1}(\boldsymbol{x}_n-\hat{\boldsymbol{x}}_{n|n-1})-\frac{1}{2}(\boldsymbol{z}_n-\boldsymbol{H}_n\boldsymbol{x}_n)^T\boldsymbol{R}_n^{-1}(\boldsymbol{z}_n-\boldsymbol{H}_n\boldsymbol{x}_n)} \tag{2.12}$$

where the product of the two constants of both distributions and the denominator of (2.9) is left out as they form the normalizing constant for the posterior and not vital for the further calculations.

As said, the MAP estimate is the point, denoted by $\hat{\boldsymbol{x}}_{n|n}$, which satisfies

$$\frac{\partial}{\partial\boldsymbol{x}_n}p(\boldsymbol{x}_n|z_{1:n}) = 0.$$

Since $\log(\cdot)$ is monotone the maximum of $\log p(\boldsymbol{x}_n|z_{1:n})$ is unchanged and it can be shown from equation (2.12) that

$$\hat{\boldsymbol{x}}_{n|n} := \big(\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n + \boldsymbol{P}_{n|n-1}^{-1}\big)^{-1}\big(\boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1} + \boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{z}_n\big),$$

by expanding, differentiating and solving for $\boldsymbol{x}_n$. The derivation steps of this equation can be found in appendix B. The matrix inversion lemma[3] simplifies the above to

$$\hat{\boldsymbol{x}}_{n|n} = \hat{\boldsymbol{x}}_{n|n-1} + \boldsymbol{K}_n(\boldsymbol{z}_n - \boldsymbol{H}_n\hat{\boldsymbol{x}}_{n|n-1})$$

[3]The matrix inverison lemma or Woodbury matrix identity states that if $\boldsymbol{M} = \boldsymbol{A} + \boldsymbol{BCD}$ then $\boldsymbol{M}^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}(\boldsymbol{C}^{-1} + \boldsymbol{DA}^{-1}\boldsymbol{B})^{-1}\boldsymbol{DA}^{-1}$, where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are matrices of appropriate size.

where the Kalman gain is defined as

$$\boldsymbol{K}_n = \boldsymbol{P}_{n|n-1}\boldsymbol{H}_n^T(\boldsymbol{H}_n\boldsymbol{P}_{n|n-1}\boldsymbol{H}_n^T + \boldsymbol{R}_n)^{-1}.$$

The prediction error is found by

$$\begin{aligned}\boldsymbol{e}_{n|n-1} &= \boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1} \\ &= \boldsymbol{F}_n\boldsymbol{x}_{n-1} + \boldsymbol{w}_n - \boldsymbol{F}_n\hat{\boldsymbol{x}}_{n-1|n-1} \\ &= \boldsymbol{F}_n(\boldsymbol{x}_{n-1} - \hat{\boldsymbol{x}}_{n-1|n-1}) + \boldsymbol{w}_n\end{aligned}$$

where $\boldsymbol{e}_{n-1|n-1} = \boldsymbol{x}_{n-1} - \hat{\boldsymbol{x}}_{n-1|n-1}$. Letting $\boldsymbol{P}_{n-1} := Cov[\boldsymbol{e}_{n-1|n-1}]$ we have

$$\begin{aligned}\boldsymbol{P}_{n|n-1} &= Cov[\boldsymbol{e}_{n|n-1}] \\ &= \boldsymbol{F}_n\boldsymbol{P}_{n-1|n-1}\boldsymbol{F}_n^T + \boldsymbol{R}_n,\end{aligned}$$

by the rules of (co)variance matrices $Cov[\boldsymbol{A}\boldsymbol{X}] = \boldsymbol{A}Cov[\boldsymbol{X}]\boldsymbol{A}^T$, where $vecX, vecY$ are stochastic vectors and $\boldsymbol{A}$ is some matrix of appropriate sizes. Furthermore,

$$\begin{aligned}\boldsymbol{e}_{n|n} &= \boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n} \\ &= \boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1} - \boldsymbol{K}_n(\boldsymbol{z}_n - \boldsymbol{H}_n\hat{\boldsymbol{x}}_{n|n-1}),\end{aligned}$$

together with $\boldsymbol{e}_{n|n-1} = \boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1}$ and $\boldsymbol{z}_n = \boldsymbol{H}_n\boldsymbol{x}_n + \boldsymbol{v}_n$, which implies

$$\begin{aligned}\boldsymbol{e}_{n|n} &= \boldsymbol{e}_{n|n-1} - \boldsymbol{K}_n(\boldsymbol{H}_n\boldsymbol{e}_{n|n-1} + \boldsymbol{v}_n) \\ &= (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n)\boldsymbol{e}_{n|n-1} - \boldsymbol{K}_n\boldsymbol{v}_n.\end{aligned}$$

By taking the covariance matrix of both side, we get

$$\begin{aligned}\boldsymbol{P}_{n|n} &= Cov[\boldsymbol{e}_{n|n}] \\ &= (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n)\boldsymbol{P}_{n|n-1}(\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n)^T + \boldsymbol{K}_n\boldsymbol{R}_n\boldsymbol{K}_n^T,\end{aligned}$$

which reduces to the final expression

$$\boldsymbol{P}_n = \boldsymbol{P}_{n|n-1} - \boldsymbol{F}_n\boldsymbol{K}_n\boldsymbol{H}_n\boldsymbol{P}_{n|n-1}.$$

This way the Kalman filter equations presented in the beginning of the chapter can be derived entirely by the MAP estimate of the Bayes filter with Gaussian noises. It should be noted, that MAP this is not the only sense in which the Kalman filter is optimal. The estimator is also the minimum mean squared error estimate. Additionally, it is also optimal in the sense that it is both an unbiased (as showed) and minimum variance estimate. [18] [16]

Chapter 3

A new model

Our goal is to construct a versatile mobility model which is flexible enough to also model environmental obstacles. Supposing that walls and obstacles exhibit a repulsive force to the movement of the device, we can attempt to construct an implementation of the Kalman filter that simultaneously models the user behaviour and builds a crude map of the environment based on the inferred positions. For this section, recall the basic setup of section 2.3 on page 19. The new version of the Kalman filter is developed in an ad-hoc and informal style.

The idea of repulsive obstacles seems plausible. First of all, users and mobile devices does not pass through obstacles. Secondly, it is a fair assumption for somewhat obvious reasons that users tend to slow down when approaching a wall. Figure 3.1 visualizes a room from above and shows the repulsive forces of the walls. Strictly, since we shall not consider the mass of the user and so the term *force* is of course used somewhat pseudo-physically. The repulsive forces can be incorporated as acceleration or velocity of the device.

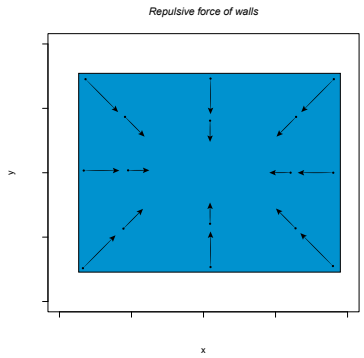


Figure 3.1: *An exaggerated visualization of the repulsive forces of the walls in some 2-dimensional room.*

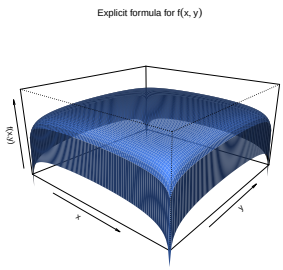


Figure 3.2: A precisely defined mathematical function which corresponds to the basic assumptions ideas. Everywhere the gradient points towards the center of the room.

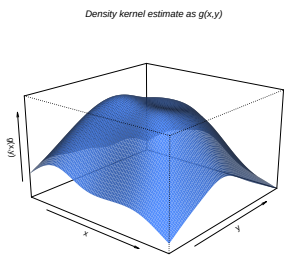


Figure 3.3: A probability density function of some inferred positions may also fulfils the basic needs. The method used here is Gaussian kernel density estimation.

Imagine a real mapping of the plane, $g : \mathbb{R}^2 \to \mathbb{R}$, of which the gradient $\nabla g(x, y)$ is a vector field describing the acceleration or velocity of the device. Recall that the gradient $\nabla g(x, y)$ points in the direction of greatest increase of g. So what would candidates for g look like? Depending on the information available, an example of g could be a precisely defined parametric function, as seen in figure 3.2, of which the parameters could be estimated from the observations. The function seen in figure 3.2 is of the form

$$g(x, y) = -\frac{1}{\sqrt[c]{(x-a)(x+a)(y-b)(y+b)}}, \qquad |x| < a,\ |y| < b,$$

which could represents a rectangular room $[-a, a] \times [-b, b]$.

Alternatively, as seen in figure 3.3, an empirical probability density function (also called kernel density estimate; a very short introduction to these can be found in Appendix A on page 42) of the inferred positions might also be a fair function for the purpose. Using the probability density function is reasonable, a least for indoor localization, since it is not too bold to claim that places frequently and rarely visited is respectively attractive and repulsive. This is consistent with the fact, that the gradient will point towards the high density regions. Areas that are never visited (such as walls and obstacles) will have low densities and thus be repulsive.

Now, let us clarify the ideas above in the 2-dimensional framework and how they can be implemented.

3.1 The 2-dimensional framework

For some preliminary analysis we use an 2-dimensional analogue to the Kalman filter presented in section 2.3 with the acceleration, and not jerk,

as the driving process noise. So, we use the simplified model described by the state model

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{w}_k$$

where $\boldsymbol{w}_k \sim N(\boldsymbol{0}, \boldsymbol{Q}_k)$, and

$$\boldsymbol{F}_k = \boldsymbol{I}_{2\times 2} \otimes \begin{bmatrix} 1 & \Delta t_k \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{x}_k = \begin{bmatrix} x_k \\ x'_k \\ y_k \\ y'_k \end{bmatrix}, \quad \boldsymbol{Q}_k = \boldsymbol{I}_{2\times 2} \otimes \begin{bmatrix} \frac{1}{4}\Delta t_k^4 & \frac{1}{2}\Delta t_k^3 \\ \frac{1}{2}\Delta t_k^3 & \Delta t_k^2 \end{bmatrix},$$

where $\otimes$ denotes the Kronecker product[1]. The observation model is given by

$$\boldsymbol{z}_k = \boldsymbol{H}\boldsymbol{x}_k + \boldsymbol{v}_k,$$

with $\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ and $\boldsymbol{v}_k = N_2(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$. Now, how do we incorporate the above ideas of using the gradient of $g(x, y)$ as contributing to the velocity of the device? One way is to interpret the gradient as a change in velocity induced by the environment. I.e. we will simply add the components of the gradient of g to the inferred speeds of $\boldsymbol{x}_k$.

To elaborate, the second and fourth components x'_k and y'_k of the state vector $\boldsymbol{x}_k$ are the intrinsic velocity of the device. Since the surroundings contribute to the velocity the perhaps easiest way to incorporate this contribution is via the control term as

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{w}_k, \tag{3.1}$$

with

$$\boldsymbol{B}_k = \begin{bmatrix} \Delta t_k & 0 \\ 1 & 0 \\ 0 & \Delta t_k \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{u}_k = \gamma \nabla g_k(x, y) = \gamma \begin{bmatrix} \frac{\partial}{\partial x} g_k(x, y) \\ \frac{\partial}{\partial y} g_k(x, y) \end{bmatrix}, \tag{3.2}$$

where γ is some scalar parameter; which could be interpreted as how much weight the environment has in the movement of the device. Note, that by this construction $\gamma = 0$ is equivalent to the regular Kalman filter with no control term. Also, by this construction, the scaled components of $\gamma \nabla f_k(x)$ enter equation (3.1) in the same manner as x'_{k-1} and y'_{k-1}. Thus, the scaled

[1] If $\boldsymbol{A}$ and $\boldsymbol{B}$ are two matrices with sizes $n \times m$ and $p \times q$ respectively then

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1m}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{n1}\boldsymbol{B} & \cdots & a_{nm}\boldsymbol{B} \end{bmatrix},$$

is the Kronecker product

gradient can rightly be interpreted as a velocity induced by earlier observations.

It can be disputed that this should be incorporated via the control when we, in reality, have no control over it. Additionally, the control vector normally represents some known forced effect to the system. The point to be noted here, is that this cannot be interpreted as a control input in the traditional sense. This implementation is simply of greatest mathematical convenience as it keeps the linearity of the system and makes the implementation very easy as the normal Kalman filter equations can be used (i.e. by choosing $\gamma = 0$).

We need also discuss how to choose appropriate functions g_k. Let us choose to use the kernel density estimate to create g_k. At time k, we have only the measurements $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ and state estimates $\hat{\boldsymbol{x}}_{1|1}, \ldots, \hat{\boldsymbol{x}}_{k-1|k-1}$ available. (We have not yet computed $\hat{\boldsymbol{x}}_{k|k}$.) Presumably, the position estimates of are better than the raw measurements and thus the set

$$\mathcal{S}_k := \left\{ \boldsymbol{H}\hat{\boldsymbol{x}}_{1|1}, \boldsymbol{H}\hat{\boldsymbol{x}}_{2|2}, \ldots, \boldsymbol{H}\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{z}_k \right\}, \tag{3.3}$$

seems the most obvious choice to use for the kernel density estimate.

If the density estimate of the inferred positions is used we need not estimate any parameters; the new modified Kalman filter procedure now looks something like the following. Supposing we are at time n and just observed z_n;

i. Compute the kernel density estimate g_k of the inferred past positions together with the new observation, i.e. the kernel density of the set $\mathcal{S}_k$, defined as equation (3.3) above.

ii. Compute $\nabla g_k(\hat{\boldsymbol{x}}_{k-1|k-1})$ and use it in $\boldsymbol{u}_k$.

iii. Compute the prediction step $\hat{\boldsymbol{x}}_{k|k-1}$ by the formulas in section 2.1.

iv. Calculate the update step $\hat{\boldsymbol{x}}_{k|n}$.

Essentially, the procedure is the Kalman filter with the extra calculation of the velocity field.

An additional perk we get from the method is that we get an rough picture of the room with the estimate density. Perhaps one can infer the placement of obstacles (and areas of open space) by low (and high) density regions. Perhaps this is a good place to repeat that by this method the device is attracted to the places already visited before and repelled by places never visited. The only way it can be said, that obstacles are taken into account is the absence of the device in these places. Indeed, the measurement error may give some observations of location in inaccessible places.

3.1.1 Performance

We use the filters on the simulated data from section 1.3. The regular Kalman filter ($\gamma = 0$) performs very well and filters out a lot of noise, cf. figure 3.4. Keep in mind that since we have simulated data the true position is known. Therefore, the the Sum of Squared Errors,

$$SSE := \sum_i \|\boldsymbol{H}\hat{\boldsymbol{x}}_{i|i} - \boldsymbol{z}_i\|^2,$$

is a good measure of performance which can be computed easily. Obviously the smaller the SSE the better. The regular Kalman filter yields a $SSE = 814.9244$ contrasted against the $SSE_{raw} = 968.6323$ of the raw measurements $\left(\sum_i \|\boldsymbol{x}_i - \boldsymbol{z}_i\|^2\right)$ for $1,554$ observations.

The best SSE that can be achieved by scaling gradient of the kernel density estimate when using the above environmental contribution to the velocity though the control vector is 814.9231; a marginally better value that the regular Kalman filter. Figure 3.6 show, perhaps alarming, the scaling-value that achieves this minima is about $\gamma = -0.04$. This is obviously somewhat contradicting to the general idea of the implementation.

Simplifying $\boldsymbol{B}_k$ of (3.2) even more by

$$\boldsymbol{B}_k = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix},$$

i.e. by adding the scaled gradient only to the velocity, yield a slightly better SEE of 814.3117 at $\gamma = -0.95$. However, in both cases the improvement is almost not noteworthy compared to the regular Kalman filter.

Plots for each state vector component in the $\gamma = 0$ Kalman filter is found in figure 3.7

3.2 Using the extended Kalman filter

There are some obvious problems with the above approach. The implementation is undesirable and informal in many ways. Using the gradient as a control vector does not reflect the normal interpretation of the control vector. A remedy is to write the gradient ∇g evaluated in $\boldsymbol{H}\hat{\boldsymbol{x}}_{k|k}$ into the transition matrix $\boldsymbol{F}_k$. However, this introduces non-linearity as $\boldsymbol{F}_k$ now depends on the position. Thus we want to implement the extended Kalman

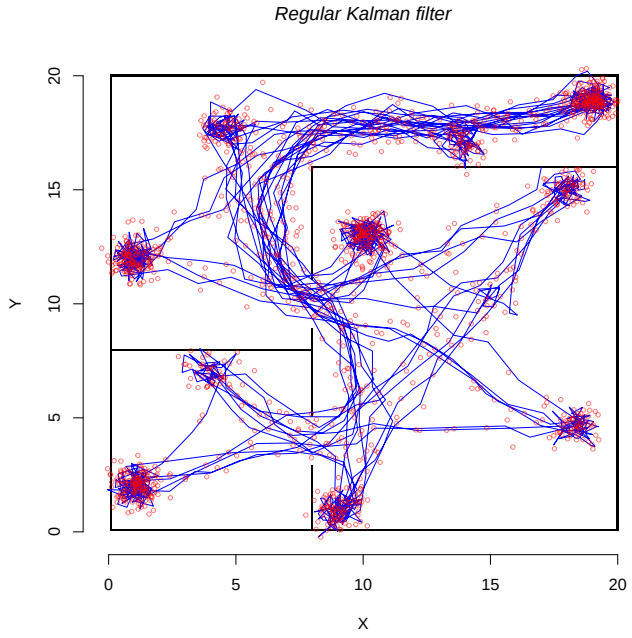


Figure 3.4: *The estimated path (blue) and noisy measurements (red) from the regular Kalman filter with no enviromental interaction.*

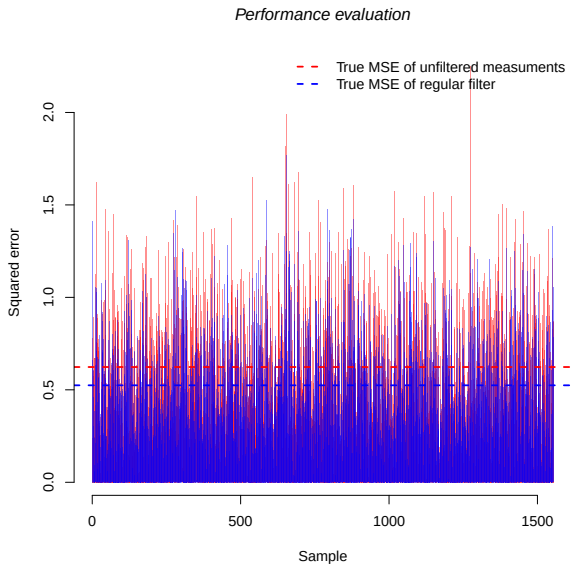


Figure 3.5: *The squared errors of each inferred position (blue) using the regular Kalman filter along with the squared error of each unfiltered measurements (red). The means of each are also plotted.*

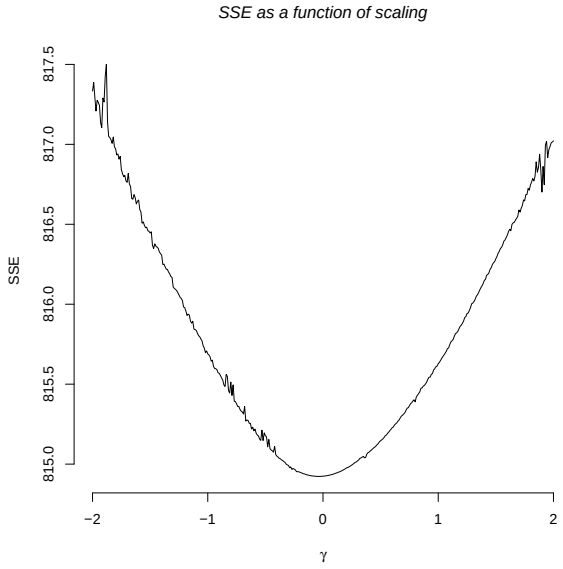


Figure 3.6: *The SSE for different values of γ which scales the gradient of the kernel density estimate. The minima is very close to zero suggesting that the regular Kalman filter should be used.*

filter in the context of equations (2.7) on page 20 with transition function

$$f(\boldsymbol{x}) = f(x, x', y, y', \gamma) = \begin{bmatrix} x + \Delta t x' + \gamma \Delta t \frac{\partial g}{\partial x}(x, y) \\ x' + \gamma \frac{\partial g}{\partial x}(x, y) \\ y + \Delta t y' + \gamma \Delta t \frac{\partial g}{\partial y}(x, y) \\ y' + \gamma \frac{\partial g}{\partial y}(x, y) \\ \gamma \end{bmatrix},$$

where the time indices k are omitted for ease of notation. The scaling parameter has now been incorporated into the state vector $\boldsymbol{x} = (x, x', y, y', \gamma)^T$. Now, obviously f is non-linear since the the kernel density function g is. The extended Kalman uses the Jacobian matrix of f which given by

$$\boldsymbol{J}_f(x, x', y, y', \gamma) = \begin{bmatrix} 1 + \gamma \Delta t \frac{\partial^2 g}{\partial x^2} & \Delta t & \gamma \Delta t \frac{\partial^2 g}{\partial y \partial x} & 0 & \Delta t \frac{\partial g}{\partial x} \\ \gamma \frac{\partial^2 g}{\partial x^2} & 1 & \gamma \frac{\partial^2 g}{\partial y \partial x} & 0 & \frac{\partial g}{\partial x} \\ \gamma \Delta t \frac{\partial^2 g}{\partial x \partial y} & 0 & 1 + \gamma \Delta t \frac{\partial^2 g}{\partial y^2} & \Delta t & \Delta t \frac{\partial g}{\partial y} \\ \gamma \frac{\partial^2 g}{\partial x \partial y} & 0 & \gamma \frac{\partial^2 g}{\partial y^2} & 1 & \frac{\partial g}{\partial y} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

with $\boldsymbol{F}_k = \boldsymbol{J}_f(x, x', y, y', \gamma)|_{(x,x',y,y',\gamma)=\hat{\boldsymbol{x}}_{k|k}}$, cf. equation (2.8) on page 21. Note, dependence of the partial derivatives on x and y has also been left out for easier notation. The function $g(x, y) = g_k(x, y)$, as stated elsewhere, is

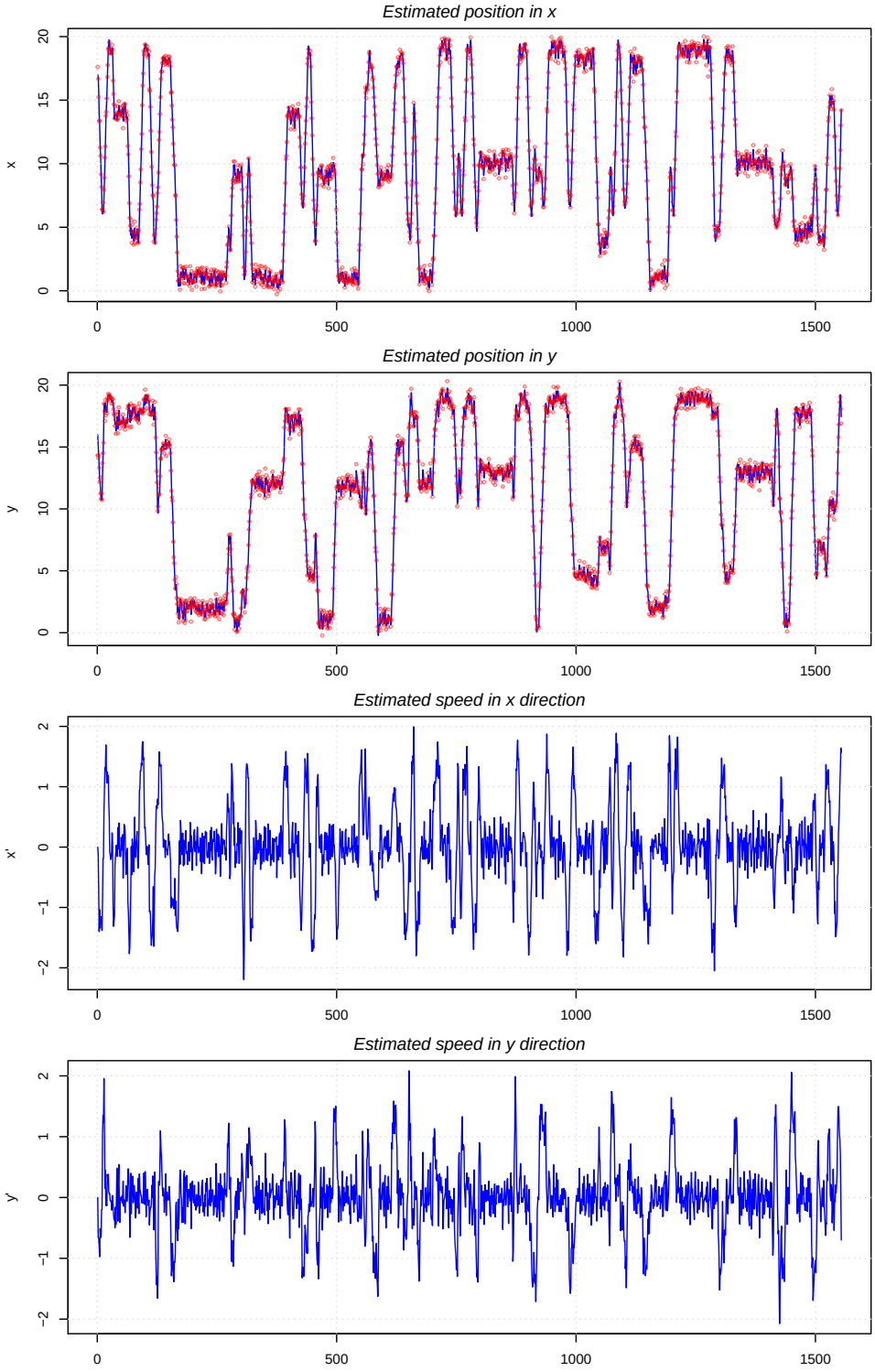


Figure 3.7: *The four components of the state vector plotted against the index with scaling $\gamma = 0$, i.e. the is regular linear Kalman filter with no control input.*

the 2 dimensional kernel density of the set

$$\mathcal{S}_k = \left\{ \boldsymbol{H}\hat{\boldsymbol{x}}_{1|1}, \boldsymbol{H}\hat{\boldsymbol{x}}_{2|2}, \ldots, \boldsymbol{H}\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{z}_k \right\}.$$

The partial derivatives in the Jacobian matrix, while they can explicitly be computed, will not be presented here. If a Gaussian kernel is used, then

$$g_k(x, y) = \frac{1}{n} \sum_{\boldsymbol{s}_i \in \mathcal{S}_k} \frac{1}{2\pi |\boldsymbol{U}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \boldsymbol{s}_i \right)^T \boldsymbol{U}^{-1} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \boldsymbol{s}_i \right) \right),$$

where $|\boldsymbol{U}|$ denotes the determinant of the 2×2 bandwidth matrix. Note, that the observation model is (almost) as before with

$$h(\boldsymbol{x}) = \boldsymbol{H}\boldsymbol{x}, \quad \text{where} \quad \boldsymbol{H} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

This concludes the construction of the extended Kalman filter.

3.2.1 Performance

The results of this extended filter is seen in figures 3.8 and 3.9. The performance of this extended Kalman filter is not as good as the linear Kalman filter with an SSE of 920.8343. Some noise has, however, been filtered out since the SSE of the observed positions was 968.6323, as stated above.

Figures 3.8 and 3.9 show some occasional very large fluctuations. These instabilities ruin overall picture of the otherwise fine performance. The parts where the extended Kalman filter behaves well, it is as good as the regular filter. For instance, the SSE computed over the indices 100 to 500 gives 206.6812 for this extended Kalman filter and 206.3436 for the regular.

Figure 3.10 shows the estimate of γ as time progress. The parameter increases from the initial value of 1 to 124.02. These increasing value, presumably large, may be explained by the tendency for the kernel density estimate to "flatten out" as the number of points increase. Alternatively it could simply be, that the extended Kalman filter diverges.

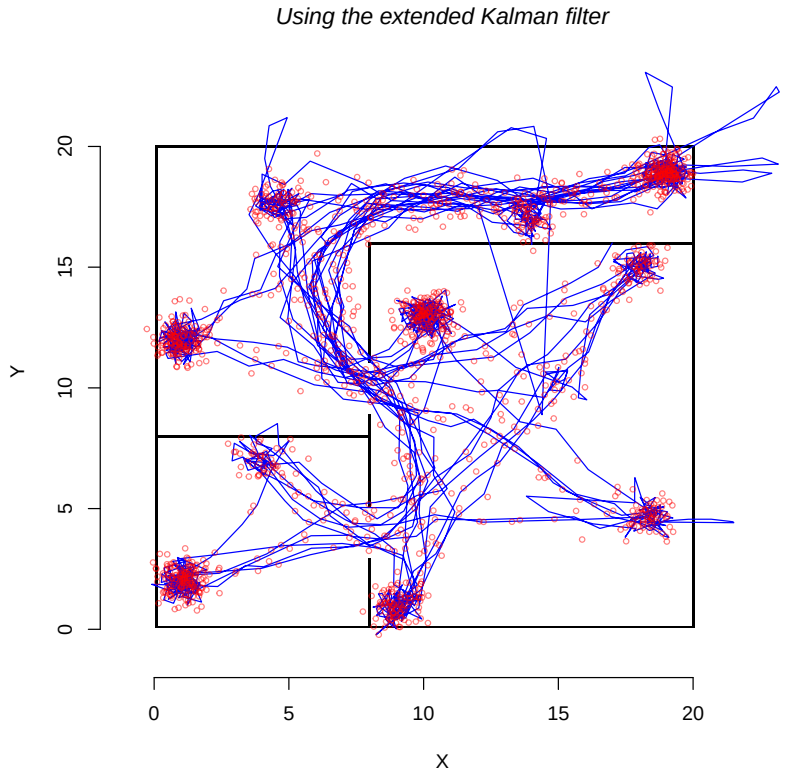


Figure 3.8: *The results of the extended Kalman filter as described. It is seen that the filter is not quite as stable and have some relatively large fluctuations. These are seen more clearly in figure 3.9.*

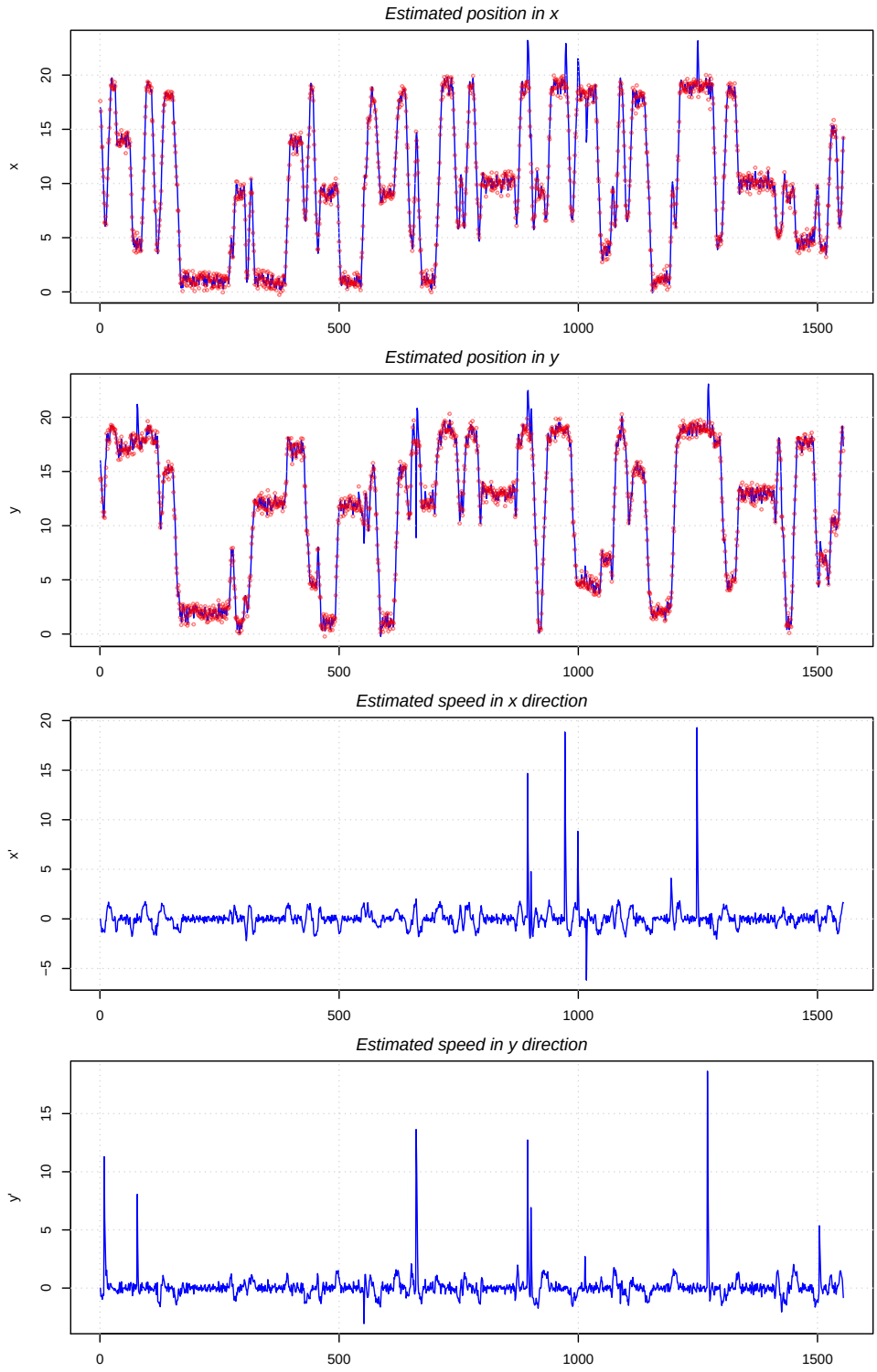


Figure 3.9: *The results of the extended Kalman filter as described. This is the first four components of the estimated state vector* $\hat{\boldsymbol{x}}_{i|i}$ *each plotted against* i*.*

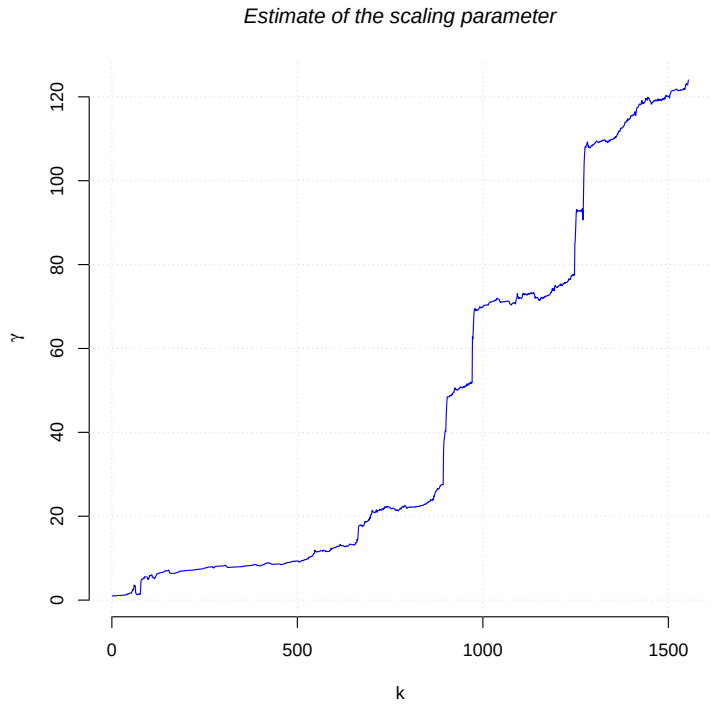


Figure 3.10: *The estimated scaling parameter γ of the extended Kalman filter as time k increases. This is the fifth component of the estimated state vector $\hat{\boldsymbol{x}}_{k|k}$ each plotted against k.*

Chapter 4

Concluding notes

That improving the Kalman filter is hard has certainly been reaffirmed. The Kalman filter is optimal and thus *cannot* be outperformed if the assumptions indeed are valid. While these proven facts obviously should come as no surprise the reality is that the assumptions almost never is perfectly satisfied; and thus there should be room for improvement.

The assumptions intuitively seem to hold approximately for the simulated data. For instance, we do know the measurement noise is perfectly simulated according to the assumptions. It is not clear what is process noise and whether it satisfies the assumptions. The abrupt starting and stopping at way-points may also contribute larger errors.

On an entirely different point, it may problematic that the basic assumptions of the dependence structure in the hidden Markov model (cf. figure 2.1 on page 22) are violated as g_k depends on all earlier inferred positions.

With these problems pointed out, it can safely be concluded that the very small gain by the control implementation is not worth the extra computational effort. The regular Kalman filter performs very satisfactory. While this heuristic approach to improve proves unfruitful there are some important lessons to be learned (or remembered).

One of the obvious strengths of the Kalman filter is that it is recursive. The kernel density estimate cannot, at least to my knowledge, be computed recursively and thus has to be recomputed every iteration. This is one obvious problem of increasing complexity of the proposed implementation.

Secondly, before applying complex non-linear filters, the linear Kalman filter (if possible) should be tested to see whether or not the extra computational effort is worth the gain (if any gain at all).

Thirdly, the extended Kalman filter can be somewhat unstable. If the perturbations can be remedied similar extended Kalman filters in the context of mobile localization might be a valid subject for further study.

While the Kalman filter is very versatile it is limited still to the linear case. The non-linear extended version is sub-optimal and can have diver-

gence problems amongst other. An alternativ filtering-scheme if the system is non-linear or the assumptions are invalid is the so-called particle filters, which relies heavily on the formulas presented in section 2.5.1 about recursive Bayesian estimation.

If the ideas are formulated more rigorously the extra estimated parameters and kernel density estimate could provide even more insight in the surroundings and behaviour of mobile users.

Bibliography

[1] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[2] V. Borrel, M. D. de Amorim, and S. Fdida, "On natural mobility models," *Lecture Notes In Computer Science*, vol. 3854.

[3] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, 2002.

[4] J.-Y. L. Boudec, "Understanding the simulation of mobility models with palm calculus," *Performance Evaluation*, no. 2007.

[5] A. Klein, "Routing protocol." http://www.routingprotokolle.de/, nov 2010. EADS Innovation Works, Dept. IW-SI - Sensors, Electronics & Systems Integration, Munich.

[6] F. Rubin, "The lee path connection algorithm," *IEEE Transactions On Computers*, 1974.

[7] V. Horak and P. Gruber, "Parallel numerical solution of 2-d heat equation."

[8] J. Kammann, M. Angermann, and B. Lami, "A new mobility model based on maps," *IEEE Vehicular Technology Conference*, 2003.

[9] R. L. Knoblauch, M. T. Pietrucha, and M. Nitzburg, "Field studies of pedestrian walking speed and start-up time," *Transportation Research Record*, no. 1538.

[10] R. L. Olsen, J. C. P. Figueiras, J. G. Rasmussen, and H. P. Schwefel, "How precise should localization be? A quantitative analysis of the impact of delay and mobility on reliability of location information," *IEEE Globecom 2010 - Communications QoS, Reliability and Modelling Symposium (GC10 - CQRM)*.

[11] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, no. 82, 1960.

[12] M. S. Greval and A. P. Andrews, *Kalman Filtering: Theory and Practice Using MATLAB*. 2nd ed., 2001.

[13] S. Haykin, *Adaptive Filter Theory*. Prentice-Hall, 4th ed., 2002.

[14] F. Orderud, "Comparison of kalman filter estimation approaches for state space models with nonlinear measurements," 2005.

[15] K. S. Shanmugan and A. M. Breipohl, *Random Signals: Detection, Estimation, and Data Analysis*. John Wiley & Sons, 1988.

[16] Z. Chen, "Bayesian filtering: From kalman filters to particle filters, and beyond," *Statistics*, 2003.

[17] P. Olofsson, *Probability, Statistics, and Stochastic Processes*. Wiley-Interscience, 2005.

[18] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," 2004.

Appendix A

Kernel density estimation

Kernel density estimation is an non-parametric approach to estimate probability density functions from which only n realisations is available. Suppose $(x_1, \ldots, x_n)$ is drawn from some density f, the kernel density estimator is then defined as

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i), \tag{A.1}$$

where $K_h(\cdot)$ is a kernel[1] with bandwith h. Figure A.2 illustrates the summation of the differently translated kernels of equation (A.1). As seen, a bandwidth of 1 and 0.1 yield too smooth and too fluctuating respectively. The choice of 0.39 is computed according to some rule-of-thumb, which shall not be described here.

Obviously, the resulting function is highly dependant on the choice of bandwidth h. The bandwidth h determines how "smooth" the estimate is, high bandwidths yield very smooth estimates while small h gives many fluctuations, as seen in figure A.3.

In two and higher dimensions the h is generalized to a matrix which contains covariances. The two dimensional kernel estimate, with bivariate Gaussian kernels, is then described by

$$\hat{f}_{\boldsymbol{H}}(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2\pi |\boldsymbol{H}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}_i)^T \boldsymbol{H}^{-1} (\boldsymbol{x} - \boldsymbol{x}_i) \right),$$

where $\boldsymbol{x}$ is a two dimensional vector and $\boldsymbol{H}$ is the bandwidth matrix.

[1] A kernel $K_h(x)$ is a function which satisfies

$$\int_{\Omega} K(x)dx = 1$$
$$K(x) = K(-x), \quad \forall x \in \Omega,$$

i.e. it integrates to one (thus is a probability distribution) and is symmetric around 0. If $K(x)$ is a kernel, then so is $K_h(x) := h^{-1}K(h^{-1}x)$. Figure A.1 shows some different kernel choices.

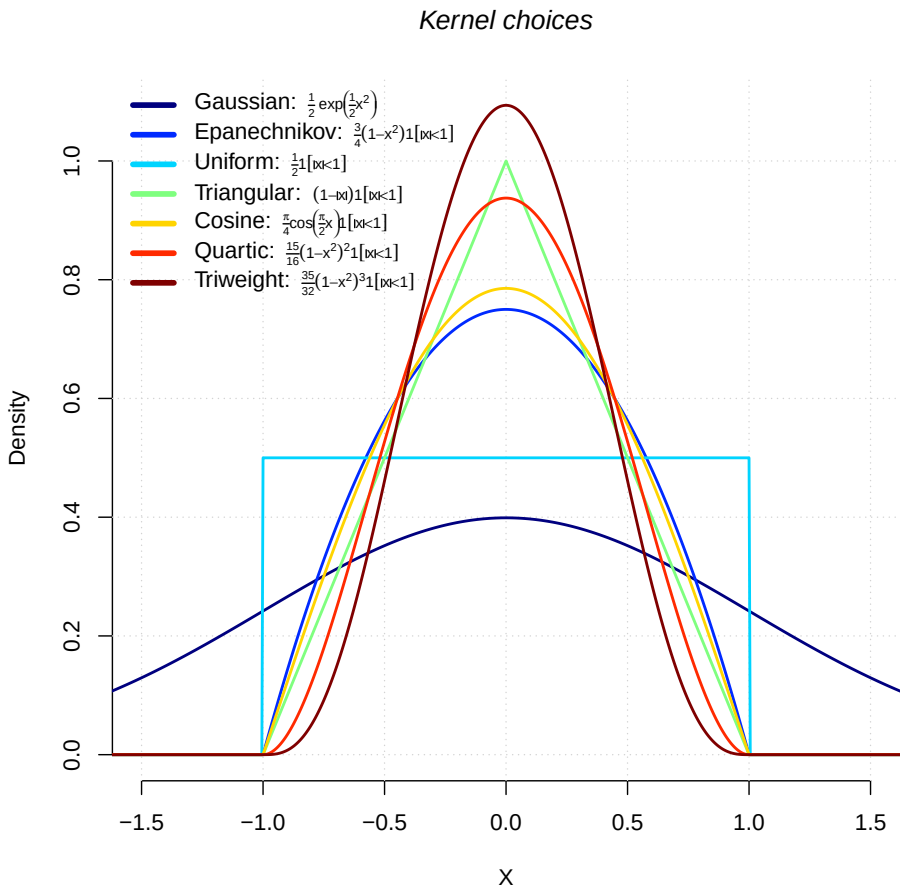


Figure A.1: *Some different popular kernels. Note, that* $\mathbf{1}[\,\cdot\,]$ *here denotes the indicator function.*

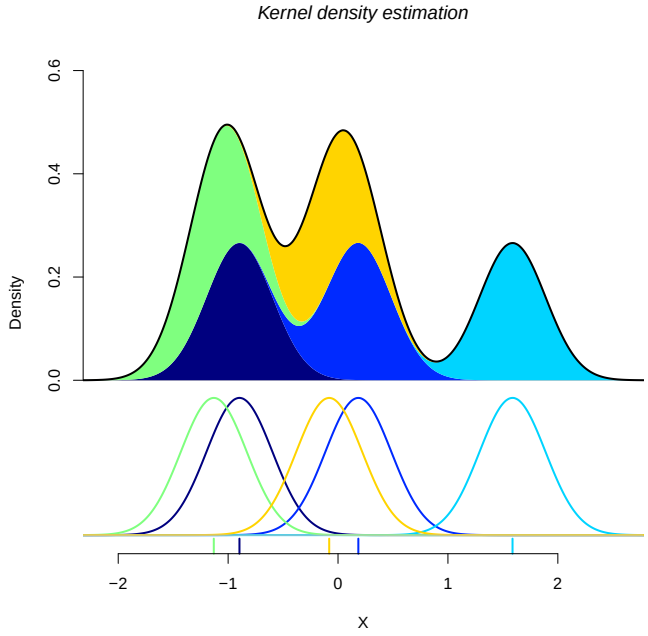


Figure A.2: *This figure illustrates the individual kernels around each realisation of f along with the summed resulting function.*

The bandwidth is selected from rules derived from some optimality criterion; the most common of which is the mean integrated squarred error given by

$$e(h) = \mathbb{E}\left[\int_{\Omega} \hat{f}_h(x) - f(x)dx\right],$$

where $\hat{h} := \operatorname{argmin}\big(e(h)\big)$. Note, that this formula cannot be applied directly as it involves the unknown function f. Figure 3.3 on page 27 shows an example of 2 dimensional kernel density estimation using Gaussian kernels.

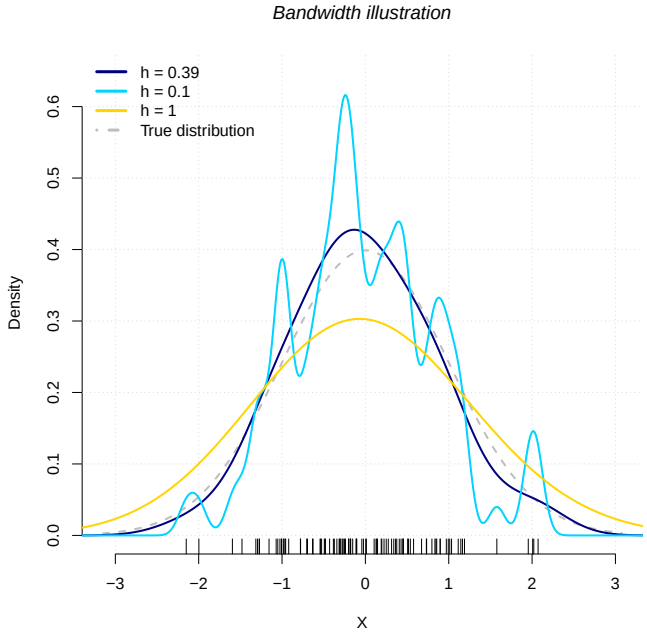


Figure A.3: *Three different kernel density estimates, all using Gaussian kernels, of 100 realisations from a standard Gaussian distribution. The data is illustrated as black ticks on the x-axis.*

Appendix B

Derivation notes

Substituting (2.12) into (2.5.2) yields

$$
\begin{aligned}
0 = \frac{\partial}{\partial \boldsymbol{x}_n}\Big(& -\frac{1}{2}(\boldsymbol{z}_n - \boldsymbol{H}_n\boldsymbol{x}_n)^T\boldsymbol{R}_n^{-1}(\boldsymbol{z}_n - \boldsymbol{H}_n\boldsymbol{x}_n) \\
& -\frac{1}{2}(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1})^T\boldsymbol{P}_{n|n-1}^{-1}(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_{n|n-1})\Big) \\
= \frac{\partial}{\partial \boldsymbol{x}_n}\Big(& -\frac{1}{2}(\boldsymbol{z}_n^T - \boldsymbol{x}_n^T\boldsymbol{H}_n^T)(\boldsymbol{R}_n^{-1}\boldsymbol{z}_n - \boldsymbol{R}_n^{-1}\boldsymbol{H}_n\boldsymbol{x}_n) \\
& -\frac{1}{2}(\boldsymbol{x}_n^T - \hat{\boldsymbol{x}}_{n|n-1}^T)(\boldsymbol{P}_{n|n-1}^{-1}\boldsymbol{x}_n - \boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1})\Big) \\
= \frac{\partial}{\partial \boldsymbol{x}_n}\Big(& -\frac{1}{2}\big(\boldsymbol{z}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{z}_n - \boldsymbol{z}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n\boldsymbol{x}_n \\
& \quad - \boldsymbol{x}_n^T\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{z}_n + \boldsymbol{x}_n^T\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n\boldsymbol{x}_n\big) \\
& -\frac{1}{2}\big(\boldsymbol{x}_n^T\boldsymbol{P}_{n|n-1}^{-1}\boldsymbol{x}_n - \boldsymbol{x}_n^T\boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1} \\
& \quad - \hat{\boldsymbol{x}}_{n|n-1}^T\boldsymbol{P}_{n|n-1}^{-1}\boldsymbol{x}_n + \hat{\boldsymbol{x}}_{n|n-1}^T\boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1}\big)\Big) \\
= \frac{\partial}{\partial \boldsymbol{x}_n}\Big(& -\frac{1}{2}\boldsymbol{z}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{z}_n + \boldsymbol{z}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n\boldsymbol{x}_n - \frac{1}{2}\boldsymbol{x}_n^T\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n\boldsymbol{x}_n \\
& -\frac{1}{2}\boldsymbol{x}_n^T\boldsymbol{P}_{n|n-1}^{-1}\boldsymbol{x}_n + \boldsymbol{x}_n^T\boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1} - \frac{1}{2}\hat{\boldsymbol{x}}_{n|n-1}^T\boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1}\Big) \\
= {} & (\boldsymbol{z}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n)^T - \boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n\boldsymbol{x}_n - \boldsymbol{P}_{n|n-1}^{-1}\boldsymbol{x}_n + \boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1} \\
= {} & \big(\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{z}_n + \boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1}\big) - \big(\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n + \boldsymbol{P}_{n|n-1}^{-1}\big)\boldsymbol{x}_n
\end{aligned}
$$

which is equivalent to

$$
\boldsymbol{x}_n = \big(\boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{H}_n + \boldsymbol{P}_{n|n-1}^{-1}\big)^{-1}\big(\boldsymbol{P}_{n|n-1}^{-1}\hat{\boldsymbol{x}}_{n|n-1} + \boldsymbol{H}_n^T\boldsymbol{R}_n^{-1}\boldsymbol{z}_n\big).
$$

When differentiating, we use the linearity of the differential operator along with the standard rules of vector calculus

$$\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A} = \boldsymbol{A}, \qquad \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T, \qquad \text{and} \qquad \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} = 2\boldsymbol{A}\boldsymbol{x},$$

where the last equality only hold if $\boldsymbol{A}$ is a symmetric matrix.

Appendix C

R-scripts

These are the essential 'stripped' programs used during the project. Most of the R work done was left out of the report. There was not enough room for the scripts that produce all the figures. Only the bare minimum is included here from which most of the above can be derived fairly easily.

Note, that some scripts may have to be executed before others.

```
##############################################################################
# Path finding with Lee algorithm                                            #
# Written by Anders Bilgrau                                                  #
# Last revisited: 26th of May, 2011                                          #
##############################################################################

set.seed(8)

#
# Constructing a layout map:
#

n <- 200
L <- matrix(1, n, n)
L[c(1,n),] <- 0
L[,c(1,n)] <- 0
L[n*0.4, 1:(n*0.8)] <- 0
L[(n*0.4):n, n*0.8] <- 0
L[1:(n*0.4), n*0.4] <- 0
L[n*0.4, c((n*0.45):(n*0.55), (n*0.15):(n*0.25))] <- 1


#
# Picking (non) random waypoints
#

# Destination
x <- 20
y <- 60

# Starting points
x.start <- c(0.70*n, 0.05*n, 0.90*n, 0.45*n, 0.05*n, 0.75*n)
y.start <- c(0.70*n, 0.75*n, 0.60*n, 0.05*n, 0.05*n, 0.95*n)
m <- length(x.start)

image(1:n, 1:n, log((L-1)*(-1)), col = "Black",
      xlab = "X", ylab = "Y", font.main = 3, axes = FALSE,
      main = "Example of a layout matrix")
axis(1); axis(2)
points(x, y, pch = 16, cex = 1, col = "Red")
points(x.start, y.start, pch = 16, cex = 1, col = "Blue")
text(x.start, y.start, paste("WP", 1:m), pos = 3, cex = 0.8)
text(x, y, "Destination.", pos = 3, cex = 0.8)


#
# Computing the matrix
#
```

```
u       <- L*matrix(0, n, n)  # 1/n^2
u[x,y] <- 1
k       <- 1

while (sum(L != 0) - sum(u > 0) != 0) {
  for (i in 2:(n-1)) {
    for (j in 2:(n-1)) {
      if (u[i,j] == 0 &
          (u[i+1, j] == k |  u[i-1, j] == k |
          u[i, j+1] == k |  u[i, j-1] == k)) {
        u[i,j] <- L[i,j]*(k + 1)
      }
    }
  }
  k <- k + 1

  if (k %% 10 == 0) {
    cat("Iteration =", k, "\n")
    cat("Missing zeros:", sum(L != 0) - sum(u > 0), "\n")
    flush.console()
  }
}
u[L == 0] <- Inf


#
# Path finding & plotting
#

pdf(file = "LeesAlgorithm.pdf")
  jet.colors <-
    colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan",
                       "#7FFF7F", "yellow", "#FF7F00", "red", "#7F0000"))
  image(1:n, 1:n, matrix(0,n,n), col = "Black",
        xlab = "X", ylab = "Y", font.main = 3,
        main = "Path finding using the Lee algortihm")
  image(1:n, 1:n, u, col = jet.colors(255), add = TRUE)
  contour(1:n, 1:n, u, add = TRUE, nlevels = 50,
          col = "#FFFFFFF0", drawlabels = FALSE)

  for (k in 1:m) {
    x.path <- rx <- x.start[k]
    y.path <- ry <- y.start[k]

    while (rx != x | ry != y) {
      uu    <- u[(rx-1):(rx+1), (ry-1):(ry+1)]
      direc <- which(uu == min(uu), arr.ind = TRUE) - 2
      rx <- rx + direc[1]
      ry <- ry + direc[2]
      x.path <- c(x.path, rx)
      y.path <- c(y.path, ry)
    }
    lines(x.path, y.path, col = "Black")
  }
  points(x,y, col = "White", pch = 16, cex = 0.8)
  points(x.start,y.start, col = "Black", pch = 16)
dev.off()
```

R-script C.1: *This script computes and produce plots of the path between points using the Lee algorithm.*

```
################################################################################
# Path finding with the heat equation                                          #
# Written by Anders Bilgrau                                                    #
# Last revisited: 26th of May, 2011                                            #
################################################################################

#
# Initializing
#

library(numDeriv)
library(akima)

par(ask = FALSE)
options(digits = 22)

#
# Constructing a layout map:
#
```

```
n <- 200
L <- matrix(1, n, n)
L[c(1,n),] <- 0
L[,c(1,n)] <- 0
L[n*0.4, 1:(n*0.8)] <- 0
L[(n*0.4):n, n*0.8] <- 0
L[1:(n*0.4), n*0.4] <- 0
L[n*0.4, c((n*0.45):(n*0.55), (n*0.15):(n*0.25))] <- 1


#
# Picking (non) random waypoints
#

# Destination
x <- 20  # sample(1:n, 1)
y <- 60  # sample(1:n, 1)

# Start
set.seed(8)
m <- 6
x.start <- c(0.70*n, 0.05*n, 0.90*n, 0.45*n, 0.05*n, 0.95*n)  # sample(1:n, m)
y.start <- c(0.85*n, 0.60*n, 0.75*n, 0.05*n, 0.10*n, 0.95*n)  # sample(1:n, m)

image(1:n, 1:n, log((L-1)*(-1)), col = "Black",
      xlab = "X", ylab = "Y", font.main = 3,
      main = "Example of a layout matrix")
points(x, y, pch = 16, cex = 1, col = "Red")
points(x.start, y.start, pch = 16, cex = 1, col = "Blue")
text(x.start, y.start, paste("WP", 1:m), pos = 3, cex = 0.8)
text(x, y, "Destination.", pos = 3, cex = 0.8)


#
# Computing the heat matrix
#

u <- L*matrix(0, n, n)  # 1/n^2
u[x,y] <- 1
uu <- matrix(0, n, n)
sims <- 3000
k <- 0.5

for (t in 1:sims) {
  for (i in 2:(n-1)) {
    for (j in 2:(n-1)) {
      uu[i,j] <-
        u[i,j] + k*((u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1])/4 - u[i,j])
    }
  }
  uu     <- L*uu
  u      <- uu    # Absorbing boundaries
  u[x,y] <- 1     # Constant temperature

  if (t %% 10 == 0) {
    cat("Iteration =", t, "\n");
    flush.console();
  }
}


#
# Example of finding of path
#

#load("HeatEquation.Rdata")

lu <- log(u)

f <- function(val) {
  rval <- round(val, 0)
  return(
    interpp(rep((rval[1]-2):(rval[1]+2), each = 5),
            rep((rval[2]-2):(rval[2]+2), 5),
            as.vector(t(lu[(rval[1]-2):(rval[1]+2),(rval[2]-2):(rval[2]+2)])),
            val[1], val[2])$z
  )
}

x.route <- rx <- x.start[1]
y.route <- ry <- y.start[1]
h       <- 0.25
```

```
eps     <- 0.5

while (abs(x - rx) > eps | abs(y - ry) > eps) {

  df <- grad(f, c(rx,ry))
  rx <- rx + h*cos(atan2(df[2],df[1]))
  ry <- ry + h*sin(atan2(df[2],df[1]))

  x.route <- c(x.route, rx)
  y.route <- c(y.route, ry)

}


#
# Plotting
#

#pdf("HeatEquation.pdf")
  jet.colors <-
    colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan",
                       "#7FFF7F", "yellow", "#FF7F00", "red", "#7F0000"))

  image(1:n, 1:n, matrix(0,n,n), col = "Black",
        xlab = "X", ylab = "Y", font.main = 3,
        main = "Path finding using the Heat equation")
  image(1:n, 1:n, log(u), col = jet.colors(255), add = TRUE)
  contour(1:n, 1:n, log(u), add = TRUE, nlevels = 150,
          col = "#FFFFFFF0", drawlabels = FALSE)
  for (j in 1:m) {
    x.route <- rx <- x.start[j]
    y.route <- ry <- y.start[j]
    while (abs(x - rx) > eps | abs(y - ry) > eps) {
      df <- grad(f, c(rx,ry))
      rx <- rx + h*cos(atan2(df[2],df[1]))
      ry <- ry + h*sin(atan2(df[2],df[1]))
      x.route <- c(x.route, rx)
      y.route <- c(y.route, ry)
    }
    lines(x.route, y.route)
  }

  points(c(x, x.start), c(y, y.start), pch = 16, cex = c(0.6,rep(1,m)),
         col = c("White", rep("Black",m)))
#dev.off()
```

R-script C.2: *This script computes and produce plots of the path between points using the Heat equation.*

```
################################################################################
# Simulation of a Indoor dataset                                               #
# Written by Anders Bilgrau                                                    #
# Last revisited: 26th of May, 2011                                            #
################################################################################

#
# Initializing
#

library(numDeriv)
library(akima)

#
# Constructing the layout map
#

size <- 20  # (meters)
n    <- 200
L    <- matrix(1, n, n)
L[c(1,n),]          <- 0
L[,c(1,n)]          <- 0
L[n*0.4, 1:(n*0.8)] <- 0
L[(n*0.4):n, n*0.8] <- 0
L[1:(n*0.4), n*0.4] <- 0
L[n*0.4, c((n*0.45):(n*0.55), (n*0.15):(n*0.25))] <- 1


#
# Picking (non) random waypoints
#
```

```
wp.x <- n*c(0.70, 0.05, 0.90, 0.45, 0.05, 0.95, 0.20, 0.50, 0.92, 0.23, 0.75)
wp.y <- n*c(0.85, 0.60, 0.75, 0.05, 0.10, 0.95, 0.35, 0.65, 0.23, 0.89, 0.50)
m    <- length(wp.x)

#
# Computing all Heat matrices
#

matrices <- list()

for (k in 1:m) {

  # Computing the heat matrix with source in (wp.x[k], wp.y[k])

  sims <- 5000
  u    <- uu <- matrix(0, n, n)
  u[wp.x[k], wp.y[k]] <- 1

  st <- proc.time()
  for (t in 1:sims) {
    for (i in 2:(n-1)) {
      for (j in 2:(n-1)) {
        uu[i,j] <-
          u[i,j] + 0.5*((u[i-1,j]+u[i+1,j]+u[i,j-1]+u[i,j+1])/4 - u[i,j])
      }
    }
    u <- L*uu                     # Absorbing boundaries
    u[wp.x[k], wp.y[k]] <- 1      # Constant temperature at destination
  }
  matrices[[k]] <- u
  cat(paste("Heatmap with source in WP", k, "of", m, "computed in",
            round((proc.time() - st)[3]%/%60), "minutes.\n")); flush.console();
  save(matrices, file = "matrices.Rdata")
}


#
# Computing all m*(m-1) possible paths
#

all.paths <- list()

for (k in 1:m) {

  u <- log(matrices[[k]])

  f <- function(v) {
    rv <- round(v, 0)
    interpp(rep((rv[1]-1):(rv[1]+1), each = 3),
            rep((rv[2]-1):(rv[2]+1), 3),
            as.vector(t(u[(rv[1]-1):(rv[1]+1),(rv[2]-1):(rv[2]+1)])),
            v[1], v[2])$z
  }

  path <- vector("list", m)

  for (i in (1:m)[-k]) {

    x.path <- rx <- wp.x[i]
    y.path <- ry <- wp.y[i]
    h      <- 0.75
    eps    <- 1

    while (sqrt((wp.x[k] - rx)^2 +(wp.y[k] - ry)^2) > eps) {
      df <- grad(f, c(rx,ry))
      rx <- rx + h*cos(atan2(df[2],df[1]))
      ry <- ry + h*sin(atan2(df[2],df[1]))
      x.path <- c(x.path, rx)
      y.path <- c(y.path, ry)
    }
    path[[i]] <- (size/n)*cbind(x = x.path, y = y.path)
  }
  all.paths[[k]] <- path
  cat(paste("All paths to WP", k, "computed.\n")); flush.console();
}


#
# Simulating walk
#

# Choosing the random waypoints
```

```
rw   <- sample(1:m, 50, replace = TRUE)
rw   <- rw[c(TRUE, rw[-1] != rw[-n.wp])]  # We walk to a new WP each time!

# Combining paths and simulating speed and waiting times

walk <- NULL

for (i in 1:(length(rw) - 1)) {
  p <- all.paths[[rw[i+1]]][[rw[i]]]
  l <- (size/n)*h*(dim(p)[1] - 1)
  s <- rnorm(1, mean = 5, sd = 0.25) * 1/3.6
  p <- cbind(p, t = seq(0, l/s, length.out = dim(p)[1]))
  wait <- rexp(1, rate = 1/20)

  if (i > 1) {
    p[, 3] <- p[, 3] + walk[dim(walk)[1], 3] + wait
  }
  walk <- rbind(walk, p)
}


#
# Simulating observation set
#

sample.rate <- 1  # samples pr. second
tmax        <- walk[dim(walk)[1],3]
ts          <- seq(0, tmax, by = 1/sample.rate)

pos <- function(t) {
  cbind(x = approxfun(walk[,3], walk[,1], yleft = NA, yright = NA,)(t),
        y = approxfun(walk[,3], walk[,2], yleft = NA, yright = NA,)(t),
        t = t)
}

std      <- 0.5  # Adding Gaussian noise
obs.path <- pos(ts)
obs.path <- cbind(obs.path, zx = obs.path[,1] + rnorm(length(ts), 0, std))
obs.path <- cbind(obs.path, zy = obs.path[,2] + rnorm(length(ts), 0, std))
obs.path <- as.data.frame(obs.path)

save(obs.path, file = "obs.path.Rdata")


#
# Real time plotting
#

real.time <- TRUE

jet.colors <-
  colorRampPalette(c("#00007F", "blue",   "#007FFF", "cyan",
                     "#7FFF7F", "yellow", "#FF7F00", "red",  "#7F0000"))
image((1:n)*size/n, (1:n)*size/n, log((L-1)*(-1)), col = "Black",
      xlab = "X", ylab = "Y", font.main = 3,
      main = "Trace of simulated walk")

lines(obs.path[,1:2], col = "Blue")
points(obs.path[,1:2], pch = 16, cex = 0.6, col = "Blue")
points(obs.path[,4:5], pch = 1, cex = .6, col = "Red")
points(wp.x*size/n, wp.y*size/n, pch = 16, cex = 1, col = jet.colors(m))
text(wp.x*size/n, wp.y*size/n, paste("WP", 1:m), pos = 3, cex = 0.8)
```

R-script C.3: *This script simulates the data presented in the introduction.*

```
################################################################################
# A indoor localization Kalman filter algorithm                                #
# Written by Anders E. Bilgrau                                                 #
# Last revisited: 26th of May, 2011                                            #
################################################################################

#
# Initialzing
#

library(numDeriv)
library(akima)
library(MASS)

#
# Implementation of the regular Kalman filter
#
```

```
use.enviroment <- FALSE
if (use.enviroment) {
  scale <- seq(-2, 2, by = 0.01)
} else {
  scale <- 0
}

sse       <- NULL
all.x.upd <- list()

st <- proc.time()

for (gamma in scale) {

  # Initalization

  x.upd <- matrix(c(17,0,16,0), 4, 1)
  x.pre <- matrix(NA, 4, 1)
  P.upd <- list(diag(4))

  xs   <- obs.path[, 1:3]
  zs   <- obs.path[, 4:5]

  std  <- 2
  n.it <- nrow(xs)

  # Running the Kalman filter

  for (i in 2:n.it) {

    dt <- xs$t[i] - xs$t[i-1]
    F  <- diag(2) %x% matrix(c(1,0,dt,1), 2, 2)
    Q  <- diag(2) %x% matrix(c(dt^4/4, dt^3/2,dt^3/2,dt^2), 2,2)
    H  <- matrix(c(1,0,0,0,0,1,0,0), 2, 4)
    R  <- diag(2)*std
    B  <- matrix(c(dt,1,0,0,0,0,dt,1), 4, 2)
    I  <- diag(4)
    u  <- matrix(0, 2, 1)

    # If use.enviroment == FALSE or scale == 0 the gradient of the
    # kernel density estimate is not used! If scale == 0, u is simply
    # zero, and it is a normal Kalman filter.

    if (use.enviroment) {
      f <- function(val, dens) {
        ix <- min(which(abs(dens$x - val[1]) == min(abs(dens$x - val[1]))))
        iy <- min(which(abs(dens$y - val[2]) == min(abs(dens$y - val[2]))))
        interpp(rep(dens$x[(ix-1):(ix+1)], each = 3),
                rep(dens$y[(iy-1):(iy+1)], 3),
                as.vector(t(dens$z[(ix-1):(ix+1),(iy-1):(iy+1)])),
                val[1], val[2])$z
      }
      set  <- cbind(x.upd[c(1,3), 1:(i-1)], t(zs[i, ]))
      dens <- kde2d(set[1, ], set[2, ], n = 50,
                    lims = c(range(set[1, ]),range(set[2, ])) + c(-1,1,-1,1))
      val  <- x.upd[c(1,3), i-1]
      u    <- gamma * matrix(grad(f, val, dens = dens), 2, 1)
    }

    # Prediction step

    x.pre <- cbind(x.pre, F %*% x.upd[, i-1] + B %*% u)
    P.pre <- F %*% P.upd[[i-1]] %*% t(F) + Q

    # Update  step

    z    <- matrix(c(zs$zx[i], zs$zy[i]), 2, 1)
    yhat <- z - H %*% x.pre[, i]
    S    <- H %*% P.pre %*% t(H) + R
    K    <- P.pre %*% t(H) %*% solve(S)

    x.upd      <- cbind(x.upd, x.pre[, i] + K %*% yhat)
    P.upd[[i]] <- (I - K %*% H) %*% P.pre

    if (i %% 100 == 0 | i == n.it | i == 2) {
      cat("Iteration", i, "of", n.it, "completed.\n"); flush.console();
    }
  }

  sse <- c(sse, sum(sqrt((x.upd[1,] - xs$x)^2 + (x.upd[3,] - xs$y)^2)))
  all.x.upd[[which(gamma == scale)]] <- x.upd
  cat("gamma =", gamma,"done\n"); flush.console();
```

```
}

#
# Performance
#

plot(scale, sse, xlab = expression(gamma), ylab = "SSE", type = "l",
     main = "SSE as a function of scaling", font.main = 3, axes = FALSE)
axis(1); axis(2);

# Measurement SSE:
m.SSE <- sum(sqrt((zs$zx - xs$x)^2 + (zs$zy - xs$y)^2))
m.SSE

# Regular Kalman filter SSE:
reg.kf.SSE <- sse[scale == 0]
reg.kf.SSE

# Best enviromental Kalman filter SSE:
env.kf.SSE <- min(sse)
env.kf.SSE
scale[which.min(sse)]

a <- proc.time() - st
a
```

R-script C.4: *This script implements and computes the results shown and discussed in chapter 3 of the new Kalman filter.*

```
################################################################################
# A indoor localization using the Extended Kalman filter algorithm             #
# Written by Anders E. Bilgrau                                                 #
# Last revisited: 26th of May, 2011                                            #
################################################################################

#
# Initialzing
#

library(numDeriv)
library(MASS)
library(akima)

#
# Auxillary funcitons
#

# These assmue the global object "dens" and "dt"


g <- function(val) {
  ix <- which.min(abs(dens$x - val[1]))
  iy <- which.min(abs(dens$y - val[2]))
  interpp(rep(dens$x[(ix-1):(ix+1)], each = 3),
          rep(dens$y[(iy-1):(iy+1)], 3),
          as.vector(t(dens$z[(ix-1):(ix+1),(iy-1):(iy+1)])),
          val[1], val[2])$z
}

dg <- function(val) {
  grad(g, val)
}

f <- function(x) { # x = (x, xp, y, yp, kappa)
  x.new  <- x[1] + dt * x[2] + dg(val = x[c(1,3)])[1] * x[5] * dt
  xp.new <- x[2]             + dg(val = x[c(1,3)])[1] * x[5]
  y.new  <- x[3] + dt * x[4] + dg(val = x[c(1,3)])[2] * x[5] * dt
  yp.new <- x[4]             + dg(val = x[c(1,3)])[2] * x[5]
  return(c(x.new, xp.new, y.new, yp.new, x[5]))
}

#
# Implementation of the Extended Kalman filter
#

# Initalization

xs <- obs.path[, 1:3]
zs <- obs.path[, c(4:5,3)]

x.upd <- matrix(c(17,0,16,0,1), 5, 1)  # Initial values
```

```
x.pre <- matrix(NA, 5, 1)
P.upd <- list(diag(5))

std  <- 2
n.it <- nrow(xs)

# Running the Kalman filter

st <- proc.time()
for (i in 2:n.it) {

  # Computing density

  set  <- cbind(x.upd[c(1,3), 1:(i-1)], t(zs[i,1:2]))
  dens <- kde2d(set[1, ],set[2, ], n = 100,
                lims = c(range(set[1,])+c(-2,2), range(set[2,])+c(-2,2)))

  # Defining matrices

  dt <- xs$t[i] - xs$t[i-1]
  F  <- jacobian(f, x = x.upd[, i-1])
  Q  <- rbind(cbind(diag(2) %x% matrix(c(dt^4/4, dt^3/2,dt^3/2,dt^2),2,2),0),
              c(0,0,0,0,1))
  H  <- matrix(c(1,0,0,0,0,1,0,0,0,0), 2, 5)
  R  <- diag(2)*std
  I  <- diag(5)

  # Prediction step

  x.pre <- cbind(x.pre, F %*% x.upd[, i-1])
  P.pre <- F %*% P.upd[[i-1]] %*% t(F) + Q

  # Update  step

  z    <- matrix(c(zs$zx[i], zs$zy[i]), 2, 1)
  yhat <- z - H %*% x.pre[, i]
  S    <- H %*% P.pre %*% t(H) + R
  K    <- P.pre %*% t(H) %*% solve(S)

  x.upd      <- cbind(x.upd, x.pre[, i] + K %*% yhat)
  P.upd[[i]] <- (I - K %*% H) %*% P.pre

  if (i %% 10 == 0 | i == n.it | i == 2) {
    time.left <- (((proc.time() - st)[3]/i)*(n.it - i))%/%60
    cat("Iteration", i, "of", n.it, "completed.\n");
    cat("Estimated time remaining:", time.left, "minutes.\n"); flush.console();
  }
}
cat("Total running time =", (proc.time() - st)[3]%/% 60)

#
# Performace
#

# Measurement SSE:
m.SSE <- sum(sqrt((zs$zx - xs$x)^2 + (zs$zy - xs$y)^2))
m.SSE

# Enviromental Kalman filter SSE:
env.SSE <- sum(sqrt((x.upd[1,] - xs$x)^2 + (x.upd[3,] - xs$y)^2))
env.SSE
x.upd[,ncol(x.upd)]
```

R-script C.5: *This script implements and computes the results of the extended Kalman filter shown and discussed in chapter 3*