Simultaneous Localization and Mapping using Extended Kalman Filter

BY:
JOHAN LEWENHAUPT

REGI:
JOSÉ A LÓPEZ OROZCO

FACULTY OF PHYSICAL SCIENCES
DEGREE IN ELECTRONIC COMMUNICATIONS ENGINEERING

Bachelor´s Degree Project
Madrid, Spain 2016

Simultaneous Localization and Mapping using Extended Kalman Filter

By:
Johan Lewenhaupt

Regi:
José A López Orozco

Faculty of Physical Sciences
Degree in Electronic Communications Engineering

UCM Universidad Complutense de Madrid
KTH Royal Institute of Technology

June 2016

Abstract

Robot localization and mapping is one of the most fundamental problems in robotics, commonly abbreviated as SLAM (Simultaneous Localization and Mapping). SLAM problems arises when a robot neither has information of its surroundings nor its of position, but by using measurement tools as well as control signals, we are able to give the robot feedback about its driving actions and the situation of the environment surrounding the robot. Given this information, the robot can estimate its position. However, the existence of uncertainty in both the driving and the measurement makes estimating the robot position a daunting task, which is handled by an Extended Kalman Filter. The Kalman Filter is a method from estimation theory that combines data of different sources and produces estimates that tend to be more accurate than a single source on its own. It is however built on linear equations, which in reality of robotics seldom arises. The Extended Kalman Filter is a version of the Kalman Filter that linearizes the equations using Taylor Series Expansion, and has become the standard in the theory of nonlinear state estimation. In this work we will firstly look at a thorough explanation of how an Extended Kalman Filter works, its advantages and disadvantages, and how it is used to estimate a position in the SLAM algorithm. Secondly, we will discuss how the SLAM algorithm works as a whole. Lastly, we will look at experiments and simulations made using MatLab.

Resumen

La localización y mapeo de robots es uno de los problemas fundamentales de la robótica, abreviado como SLAM (Simultaneous Localization and Mapping). Problemas de SLAM surgen cuando un robot no tiene información de su entorno ni su de posición, pero mediante el uso de instrumentos de medición, así como de las señales de control, se es capaz de obtener realimentación sobre sus acciones de conducción y sobre el entorno que rodea al robot y dada esta información, el robot puede calcular su posición. Sin embargo, la existencia de incertidumbre tanto en la conducción como en la estimación del entorno hace de la estima de la posición del robot una tarea de enormes proporciones, que puede ser manejado por un Filtro de Kalman Extendido. El Filtro de Kalman es un método de la teoría de estimación que combina datos de diferentes fuentes y produce estimaciones que tienden a ser más precisas que una sola fuente por sí sola. Sin embargo, se basa en ecuaciones lineales, que en la realidad de la robótica rara vez se presenta. El Filtro de Kalman Extendido es una versión del filtro de Kalman que linealiza las ecuaciones usando el desarrollo en serie de Taylor, y se ha convertido en un estándar en la teoría de estimación de estado no lineal. En este trabajo, en primer lugar, vamos a ver una explicación completa de cómo funciona un Filtro de Kalman Extendido, sus ventajas y desventajas, y cómo se utiliza para estimar una posición en el algoritmo de SLAM. En segundo lugar, vamos a debatir sobre el funcionamiento del algoritmo de SLAM en su conjunto. Por último, vamos a ver los experimentos y simulaciones realizadas utilizando MatLab.

Preface

The project

This thesis is the results of the project I performed for the bachelor's degree in Engineering Physics for Royal Institute of Technology (KTH), Stockholm, but conducted during an Erasmus semester at Universidad Complutense de Madrid (UCM), Madrid. In this thesis I look a part of the large field robotics, more specifically robot localization. This is a major field within robotics and has multiple practical purposes and uses within robotics. This topic is today very relevant, and is responsible for many upcoming technologies. A good example of an upcoming technology is self-driven cars.

Audience

The audience intended to read this thesis are students considering to enter the field of robotics and more specifically students wishing to study in-depth robotic state estimation. This would then act as an overview of some of the most important topics within the field. The courses required to read this report are linear algebra, calculus and understanding of numerical methods using MatLab. A course in control engineering could also be helpful.

Table of Contents

Simultaneous Localization and Mapping using Extended Kalman Filter 1
Abstract 3
Resumen 4
1. Introduction 7
1.1 Objectives 7
1.1.1 KF, EKF and SLAM 7
1.1.2 MatLab implementation 8
1.1.3 Results 8
1.2 Autonomous Robots 8
1.2.1 Having Mobility 8
1.2.2 Dynamic Model of a Robot 9
1.2.2 Sensing the environment 9
1.2.3 Model of a Sonar 9
1.3 Overview of the report 10
2. Estimating Position 11
2.1 The Kalman Filter 11
2.2 The Extended Kalman Filter 14
2.3 Simultaneous Localization and Mapping 17
3. MatLab Implementation 19
3.1 Description of the Map 19
3.2 Description of the Routes 21
3.3 Description of the Robot 22
3.4 Dynamic Model 22
3.5 SLAM Algorithm 23
3.6 Illustration of Results 23
4. Results and Discussion 26
4.1 Algorithms used in different Examples 26
4.2 Discussion 30
5. Conclusion and Future Work 38
6. Bibliography 39

1. Introduction

Imagine yourself in an unknown area. It is filled with objects, such as houses, cars and trees. You start walking, to find your way home. At every moment, your eyes tell you where you are, and constantly correct the imperfections of our steps. To illustrate the importance of this correction, try walking in a straight line with your eyes closed. You will most likely feel as if you are walking in the wrong direction, and might even loose balance. Your eyes will give you information about your surroundings, perhaps you see a metro leading home, or even a place you recognize that can tell you where you are.

Now imagine a robot. It will have precisely the same problem. Even though we send it in a straight path, stones and cracks on the ground could change its path when driven over. Even something as imperfections of the robot could the path. For example, perhaps one wheel on a two-wheeled robot is slightly smaller than the other one, and it therefore steers slight off all the time.

This is what SLAM is all about. A robot could use a measurement tool instead of an eye, and would use that to correct the imperfections of its movements. If it would see a recognizable object, or landmark, it can use that to correct its movements and even place itself within the discovered area.

1.1 Objectives

The main objective of his bachelor's thesis is to study and learn how a classical algorithm works and is used in robotics. This objective can be divided into three sub-objectives: studying the three different estimation methods (KF, EKF and SLAM), implementing the algorithms in MatLab to deepen the knowledge of how they work, to show the results of the program and to finally discuss its main properties.

1.1.1 KF, EKF and SLAM

To study the Kalman Filter (KF) algorithm, the Extended Kalman Filter (EKF) algorithm and the Simultaneous Localization and Mapping (SLAM) algorithm. Learn the differences of the three, the different advantages and disadvantages.

1.1.2 MatLab implementation

Create a program using MatLab that creates a virtual robot and implements the SLAM method using an Extended Kalman Filter. To use the method, the robot must also have some sort of measurement method. The robot also need a map to drive upon, which is to be created.

It is important to note that the map and the robot will be a 2-dimensional configuration.

1.1.3 Results

Show results using a few examples of both EKF and SLAM, and illustrate the errors as well as mappings. Calculate the difference between with and without landmarks, to see the influence of the algorithm.

1.2 Autonomous Robots

Autonomy used in robotics means to have an unmanned robot. A high degree of autonomy means to have a robot that performs tasks without human interference. This chapter addresses the different autonomous abilities necessary for a robot to perform the tasks necessary to execute a SLAM program.

1.2.1 Having Mobility

The mobility of a robot is the extent of how freely it is able to move its joints and its body. For example, a long bus without a bending point in the middle has less mobility than a long bus with a bending point. If we want robots that complete everyday tasks like cleaning the floor on its own, as well as for example dusting of shelfs, autonomous mobility is a key factor.

The robot to be used in my program is a small rectangular robot with two wheels. It is not optimal, serves to complete all of its tasks. There are many examples of more advanced robots that for example mimic movements of animals to increase its mobility. A good example is a robot made by Boston Dynamics (acquired by Google) called “Big Dog” *.

When constructing a robot, it needs some sort of dynamic model, making it move and turn in the desired direction. This model will depend on the robot used.

* Article about Big Dog Boston Dynamics: http://techcrunch.com/2013/12/14/google-buys-boston-dynamics-creator-of-big-dog/

1.2.2 Dynamic Model of a Robot

Having a mobility causes the need of having a model of its dynamics, since we wish to study and estimate its movements. This dynamic model can change drastically based on what type of movement the robot is able to make. For example, a two wheeled robot operating on a planar surface (which is what my program uses), would have a model like the following:

$$\dot{x} = v\cos(\theta),\ \dot{y} = v\sin(\theta),\ \dot{\theta} = \omega$$

This model says that the robot would have a speed v and an angular speed ω. The x and y position are the given through the above formula, and the turning is given by the rotational angle. Therefore, we have everything we need from this model, which is to simply move in a 2-dimensional map.

1.2.2 Sensing the environment

Robot navigation is the mission of an autonomous robot to move securely from one location to another. This requires the ability of being able to know its surroundings in some way. There are many ways to go about this problem, whereas an easy way is to use a sonar. If an object is nearby, the sonar would notice it and let the robot know. More advanced robots could use a video camera to get a real-time view, sort of like a human eye. This requires an immense amount of computational power, since image processing is usually a great task, and a very advanced program that is able to calculate distance of seen objects. The major benefits could be that the robot would be able to see more precisely how its world really is.

My MatLab implementation will use a virtual sonar, and thus, we require a model of that sonar.

1.2.3 Model of a Sonar

A sonar is a technique using sound to measure distance. It works by first sending out a pulse, and clocking the time it takes for it to come back. If we know the speed of sound based on different mediums the sound travels through, we know can calculate the distance with the following simple formula:

$$s = \frac{v \cdot t}{2}.$$

It is divided by two because of the fact that the sound has to travel fourth and back.

My MatLab simulation will simulate being run in an environment filled with air, and therefore the sonar is able to calculate distance to a point. By rotating the sonar around the robot, it can calculate the distance to all of its surroundings. However, since sound naturally spreads, the results will become imprecise if we allow the sonar to calculate distances that are too far way. Therefore, we have a limit on the sonar's reach.

1.3 Relation to my Degree

Before this project, I took a several courses that helped me understand the theory regarding estimation theory, as well as to be able to write MatLab code. The most important ones were: Calculus, Linear Algebra, Numerical Methods, Control Theory, Probability Theory and Statistics, and also Mechanics. This project will conclude my Bachelor Degree, and lead me into a Master's Degree in Robotics and Control Theory.

1.4 Overview of the report

In the following chapters I will describe how one can use the three different algorithms (KF, EKF, SLAM) to estimate a position of a robot. I will discuss how to implement the algorithms as well as a virtual robot into MatLab. Lastly, I will visualize the results and discuss what they were.

The table of contents should provide a basic overview of this report. Other than that, it could also be good to know that this report includes a lot of pictures to illustrate results, and below the picture is a short, formal description of what the picture tells. The surround text of the picture will also describe its meaning. Other than pictures, a lot of mathematical formulas and texts are included and will be explained by its surrounding text. Especially the three major algorithms of this report (KF, EKF and SLAM) will be found explained considerably.

2.Estimating Position

As mentioned, there are several ways of estimating robot position. In this report, I am going to discuss three algorithms used to estimate a robot position based on measurements of the exterior. The theory underlying these algorithms are explained in this chapter.

2.1 State of the art

Estimation theory began with the well-known method of least squares. The method was accredited to *Carl Friedrich Gauss* (in 1795), in order to solve non-linear problems in Astronomy. More specifically, how to describe planet and comet motion. Since then, several great mathematicians have been working on estimation theory in all kinds of fields, and at around 1960, Rudolf E. Kálmán created the Kalman Filter, which would come to be revolutionary in signal processing, control systems and guidance, navigation and control. Although a similar algorithm was created earlier by Thorvald Nicolai Thiele and Peter Swerling, the method got accredited to Kálmán when he visited NASA, where the team of the Apollo program saw the potential in his work. They wanted to use his algorithm for non-linear problems regarding trajectory estimation. The only issue was that the Kalman Filter could only be used for linear equations. This lead to research of an extension of the algorithm, that could be used for non-linear equations, hence the Extended Kalman Filter (published 1961). This new algorithm was developed by NASA Ames (a major research center of NASA), using calculus to linearize the equations at hand. It wasn't until the 1980s that SLAM was developed, by Smith and Durant-White, which was a major step in the field of robotics. Finally, we could model the so called "chicken and egg" problem, of trying to understand the surroundings of a robot while estimating its own position at the same time.

2.2 The Kalman Filter

The Kalman Filter algorithm is an algorithm that uses a sequence of measurements observed over time, known control inputs and a system's dynamics model. Based on these variables, it creates an estimate of the system's future state.

The Kalman Filter represents beliefs at each moment in time. At time t, the belief is represented by the mean μ_t and the covariance Σ_t. The next state probability, $p(x_t|u_t, x_{t-1})$, is a linear function with added Gaussian noise. This is expressed by the following equation:

$$x_t = A_t x_t + B_t u_t + \varepsilon_t .$$

Here, x_t and x_{t-1} are the state vectors, and u_t the control vector at time t. Both of these vectors are vertical vectors, that is, they are of the form

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix} \text{ and } u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{m,t} \end{pmatrix}.$$

A_t and B_t are matrices. A_t is a square matrix of the size $n \times n$, where *n* is the dimension of the state vector. B_t is of size $n \times m$, with *m* being the dimension of the control vector. By multiplying the state and control vector with the matrices A_t and B_t, respectively, the state transition function becomes linear. This is an important aspect: Kalman Filters only accept linear system dynamics. The random variable ε_t is a Gaussian random vector that models the white noise in the state transition. It is of the same dimension as the state vector. Its mean is zero and its covariance will be represented by R_t.

We also incorporate a measurement probability $p(z_t | x_t)$, that also must be linear. z_t is the measurement vector that is given by:

$$z_t = C_t x_t + \delta_t .$$

C_t is a matrix of size $k \times n$, where *k* is the dimension of the measurement vector. The vector δ_t describes measurement noise, with a mean of zero and covariance Q_t. The linear matrices *A*, *B* and *C* depends on the model of the robot. For a two wheeled robot, they are found in the following way:

$$A = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$$
$$B = \begin{pmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{pmatrix} \quad .$$
$$C = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

A and *B* multiplied with the state vectors:

$$\bar{x} = \begin{pmatrix} x_{position,t} \\ v_t \end{pmatrix}, \ \bar{u} = a,$$

where $x_{position}$ is a vector containing the three state variables *x*, *y* and θ, v is the speed and a is the acceleration of the robot. This gives us the following:

$$A\bar{x} = \begin{pmatrix} 1 & -\Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_{position} \\ v \end{pmatrix} = \begin{pmatrix} x_{position} + \Delta t \cdot v \\ v \end{pmatrix}$$

$$B\bar{x} = \begin{pmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{pmatrix} a = \begin{pmatrix} \frac{\Delta t^2}{2} \cdot a \\ \Delta t \cdot a \end{pmatrix} \quad .$$

$$\Rightarrow \overline{x_{t+1}} = \begin{pmatrix} x_{position,t} + \Delta t \cdot v_t \\ v_t \end{pmatrix} + \begin{pmatrix} \frac{\Delta t^2}{2} \cdot a_t \\ \Delta t \cdot a_t \end{pmatrix} = \begin{pmatrix} x_{position,t} + \Delta t \cdot v_t + \frac{1}{2} a_t \Delta t^2 \\ v_t + \Delta t \cdot a_t \end{pmatrix}$$

One immediately sees that the dimensions works out, with a position in the first column, and a speed in the second column. The matrix *C* will do nothing more than to convert the error into a position error only, by multiplying it with 1 in the first column, and 0 in the second column. This gives the updated estimated position, with a robot as an example.

We are now ready to present the classical Kalman Filter algorithm, as you find below:

1. $Inputs: \mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
2. $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3. $\overline{\Sigma_t} = A_t \Sigma_{t-1} A_t^T + R_t$
4. $K_t = \overline{\Sigma_t} C_t^T \left(C_t \overline{\Sigma_t} C_t^T + Q_t \right)^{-1}$
5. $\mu_t = \bar{\mu}_t + K_t \left(z_t - C_t \bar{\mu}_t \right)$
6. $\Sigma_t = \left(I - K_t C_t \right) \overline{\Sigma_t}$
7. $return: \mu_t, \Sigma_t$

Line number 1 simply shows the inputs of the algorithm. As mentioned before, it's the last estimated state μ_{t-1}, the covariance Σ_{t-1}, the control signal u_t, and lastly the measurement vector z_t. These variables are going to be updated and used to run the program again.

Lines number 2 and 3 creates estimates, or beliefs, at time t of the mean and the covariance of the state, based on the linear model given by the matrices *A* and *B*.

Line number 4 computes the Kalman gain K_t, which is a quantity of how much the measurement is to be incorporated into the new state.

Line number 5 is a very important line. It computes the updated position based on the Kalman gain times the error, which is the real sensor measurement, z_t, minus the sensor model (predicted measurement), $C\mu_t$.

Line number 6 creates the update for the covariance by simply taking the old covariance and reducing it by the old times the Kalman gain $\left(\Sigma_t - K_t C_t \Sigma_t = (I - K_t C_t)\Sigma_t\right)$.

Finally, line number 7 returns the new estimated state of the robot position. The robot would then continue moving and return this algorithm with a last computed mean and covariance as the new inputs, along with a new measurement.

Essentially, lines 4-6 is the whole Kalman process. This is where the correction happens based on a measurement. Without those lines, it would be a simple estimation model (lines 2-3) using dynamics, that would not account for any error at all.

2.2 The Extended Kalman Filter

An Extended Kalman Filter works with non-linear systems as well, as mentioned before, invented by NASA Ames research center to be able to implement Kalman Filters into more engineering problems.

An EKF works similarly like the Kalman Filter. Instead of using linear matrices to compute an estimate, the EKF uses nonlinear functions as stated below:

$$\begin{aligned} x_t &= g(u_t, x_{t-1}) + \varepsilon_t \\ z_t &= h(x_t) + \delta_t \end{aligned} .$$

A model like this generalizes the linear model of the Kalman Filter. The non-linear function *g* replaces the matrices A_t and B_t, and the matrix C_t is replaced with the non-linear function *h*. These two functions are later linearized with a method called *first order Taylor Expansion* learned from calculus. Starting with *g*, showed underneath:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}).$$

Here μ_t is the estimated state at time *t*. We also denote $g'(u_t, \mu_{t-1}) = G_t$ and receive the following:

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}).$$

This is a linear approximation of the function *g*. Let's start by describing how $g(u_t, \mu_{t-1})$ is found. $g(u_t, \mu_{t-1})$ is obtained from $g(u_t, x_{t-1})$:

$$g(u_t, x_{t-1}) = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\theta + \frac{v_t}{\omega_t}\sin(\theta + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\theta - \frac{v_t}{\omega_t}\cos(\theta + \omega_t \Delta t) \\ \omega \Delta t \end{pmatrix}$$

By replacing the exact state x_{t-1} that we don't know, by the estimated state μ_{t-1}, that we do know, since it is our input. We then receive the $g(u_t, \mu_{t-1})$, shown below:

$$g(u_t, \mu_{t-1}) = \mu_{t-1} + \begin{pmatrix} -\frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} + \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ \omega \Delta t \end{pmatrix}.$$

Now let's explain how G_t is found. G_t is the derivative of $g'(u_t, \mu_{t-1})$, with respect to μ_{t-1}. Therefore, G_t is found in the following way:

$$G_t = g'(u_t, \mu_{t-1}) = \begin{pmatrix} 1 & 0 & \frac{v_t}{\omega_t}\cos\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\cos(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 1 & \frac{v_t}{\omega_t}\sin\mu_{t-1,\theta} - \frac{v_t}{\omega_t}\sin(\mu_{t-1,\theta} + \omega_t \Delta t) \\ 0 & 0 & 0 \end{pmatrix}.$$

Now, the same concept is applied for *h*:

$$h(x_t) \approx h(\mu_t) + h'(\mu_t)(x_t - \mu_t) = h(\mu_t) + H_t(x_t - \mu_t),$$

Where we used $h'(\mu_t)=H_t$. Again, let's explain how both $h(\mu_t)$ and H_t are found, starting with $h(\mu_t)$. $h(\mu_t)$, the sensor model, is found in the following way:

$$h(\mu_t)=\begin{pmatrix}\sqrt{(m_{j,x}-\mu_{t,x})^2+(m_{j,y}-\mu_{t,y})^2}\\ \arctan 2(m_{j,y}-\mu_{t,y},m_{j,x}-\mu_{t,x})-\mu_{t,\theta}\\ m_{j,s}\end{pmatrix}.$$

Here, $(m_{j,x}\ m_{j,y})^T$ are the coordinates of the *j*-th landmarks observed at time *t*. $m_{j,s}$ is the signature of the landmark, if there are different kinds, which there will not be in my examples.

Now on to discover out how H_t is found. Again, H_t is the derivative of $h(\mu_t)$ and is therefore found just as in the case with G_t. Below is the calculation of the derivative:

$$H_t=h'(\mu_t)=\begin{pmatrix}\frac{m_{j,x}-\mu_{t,x}}{\sqrt{q_t}} & \frac{\mu_{t-1,y}-\mu_{t,y}}{\sqrt{q_t}} & 0\\ \frac{\mu_{t,y}-\mu_{t-1,y}}{q_t} & \frac{m_{j,x}-\mu_{t,x}}{q_t} & -1\\ 0 & 0 & 0\end{pmatrix}.$$

Here, $q_t=(m_{j,x}-\mu_{t,x})^2+(m_{j,y}-\mu_{t,x})^2$. Thus, we are done, and are again ready to present the algorithm, this time of the Extended Kalman Filter. It is showed in the following mathematical text:

1. $Inputs:\mu_{t-1},\Sigma_{t-1},u_t,z_t$
2. $\bar{\mu}_t=g(u_t,\mu_{t-1})$
3. $\bar{\Sigma}_t=G_t\Sigma_{t-1}G_t^T+R_t$
4. $K_t=\bar{\Sigma}_t H_t^T\left(H_t\bar{\Sigma}_t H_t^T+Q_t\right)^{-1}$
5. $\mu_t=\bar{\mu}_t+K_t\left(z_t-h(\bar{\mu}_t)\right)$
6. $\Sigma_t=(I-K_tH_t)\bar{\Sigma}_t$
7. $return:\mu_t,\Sigma_t$

As you can see, it is very similar to the Kalman Filter algorithm, with one major difference. The usage of a non-linear function in line 2, and usage of linearized matrices in lines 2-6. However, a line for line explanation will still be given.

Line 1 is as before the inputs. We use the estimated state μ_{t-1}, the covariance Σ_{t-1}, the control signal u_t, and lastly the measurement vector z_t.

Line 2 creates an estimate of the state, using the non-linear function $g(u_t, \mu_{t-1})$, and line 3 creates an estimate of the covariance, using the linear function G_t.

Line 4, as with the Kalman Filter, creates the Kalman gain, which, as already mentioned, is a measure of how much the new measurement is to be incorporated into the new state.

Line 5 is the update step of the state, and uses the old state + the Kalman gain times the error. The error is the subtraction of z_t, the real sensor measurement, and $h(\bar{\mu}_t)$, the sensor model.

Line 6 works just the same as with the Kalman Filter. The only difference is the usage of the linear function H_t instead of *C* as in the previous case.

Now we will move on to the SLAM algorithm in the upcoming section.

2.3 Simultaneous Localization and Mapping

SLAM problems arise whenever a robot neither has information of the environment and its own pose within it. As the name of the algorithm implies, the goal is to acquire a map of its environment and localize itself within it, simultaneously.

There are two main forms of SLAM, the *online SLAM*, and the *full SLAM*. The only difference between the two revolves around the fact that the online SLAM algorithm obtains the map and the current robot pose, whereas full SLAM obtains the map and the robot's entire path. In this report, only online SLAM will be covered.

As mentioned earlier, SLAM uses an Extended Kalman Filter to correct its pose based on measurements. What's new is to also save the measurements and include them into the state vector. The algorithm is shown on the next page.

1. Inputs : $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$
2. $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3. $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4. for all detected landmarks :
5. $z_t^i = h(\mu_t, M_t)$
6. if new landmark :
7. insert landmark's data into state vector and covariance
8. elseif :
9. $A_t^i = z_t^i - h(\bar{\mu}_t, 0)$
10. $K_t^i = \bar{\Sigma}_t H_t^{iT} \left(H_t \bar{\Sigma}_t H_t^{iT} + Q_t \right)^{-1}$
11. $\mu_t = \bar{\mu}_t + \sum_i K_t^i A_t^i$
12. $\Sigma_t = \left(I - \sum_i K_t^i H_t^i \right) \bar{\Sigma}_t$
13. end
14. end
15. return : μ_t, Σ_t

Lines number 1-3 is the same of the Extended Kalman Filter algorithm. However, at line 4 we have our first difference. Line 4 creates a loop of the lines 5-14, one loop for each landmark found in the most recent measurement. Line number 5 creates a measurement vector z_t, using the non-linear model just like in the EKF. Line number 6 determines if the landmark is newly found, and if so, line number 7 includes its position and covariance into the state-vector and the covariance respectively. If an observed landmark is already found, it will activate lines 9-12, that create a correction of the robots position its covariance. Finally, at the end (line 15), we return the values, move the robot, create a new measurement, and rerun the algorithm.

Since the state-vector and covariance in the SLAM algorithm always grows bigger for each landmark, too many landmarks make the program very heavy computational wise.

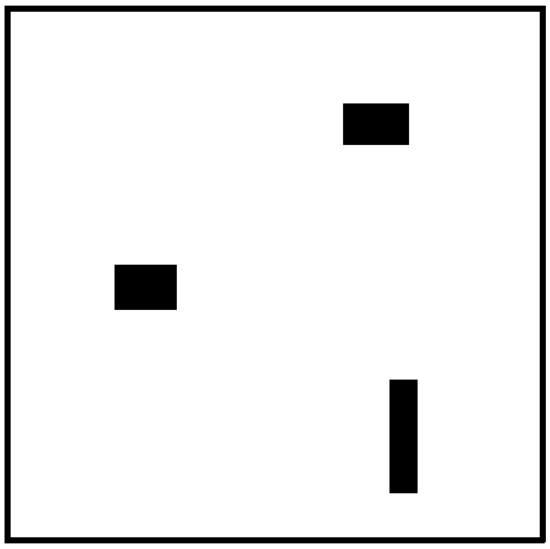
It should also be noted that the update step (lines 9-12) is only activated once the robot sees a landmark that is has already detected. Only then can it calculate an improvement of the estimate, based on his movements since seeing it last time. This means that if the robot only sees all the landmarks it detects once, the update step will never occur, and we basically have an error that always grows. In this case, the algorithm is the same as the EKF without an update, except for the fact that the state vector has grown bigger for each landmark found.

3.MatLab Implementation

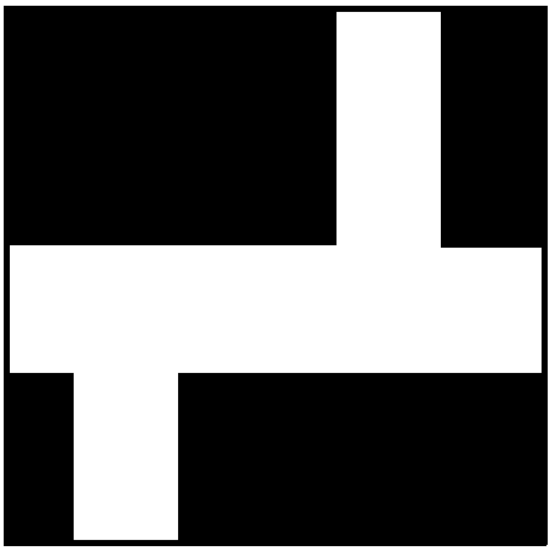
MatLab (Matrix Laboratory) is a numerical computing programming language. Developed by Cleve Moler, chairman at University of the computer science department at Mexico, in the late 1970s. It serves our task of showing the SLAM algorithm very well, and is therefore used. This chapter describes in an overall way how the program works. For acquiring details, simply view the belonging MatLab files.

3.1 Description of the Map

To begin with, we need realistic maps for a robot to run in. Created by Microsoft paint, the two following maps are used. They could portray for example a living room, or a street passage. For our task it suffices with these two maps.



Picture 1: Map 1

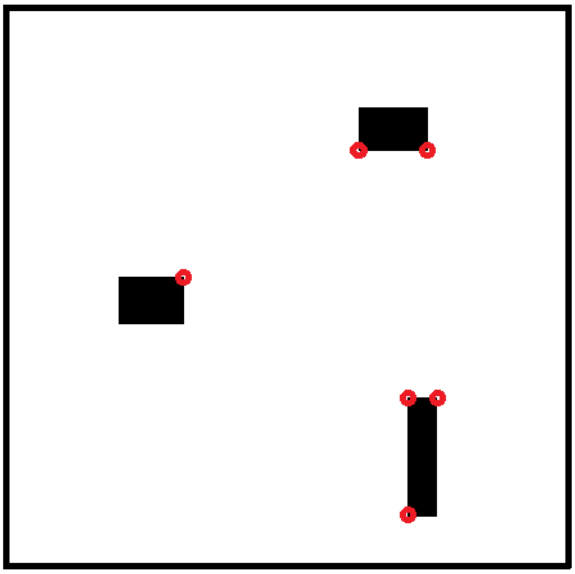


Picture 2: Map 2

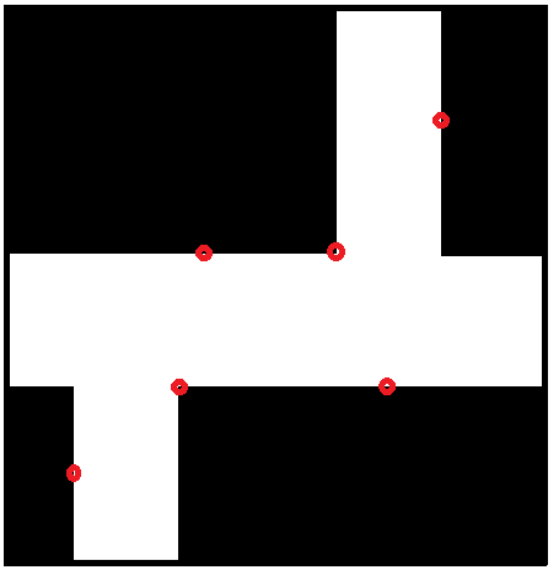
In these pictures, the white areas are open fields, and black areas are walls. The robot can thus move freely within the white areas. However, as will soon be shown, premade routes are to be constructed. This will be handled in an upcoming section.

Moreover, we need to place specific landmarks, or beacons, in these maps. As mentioned earlier, SLAM with too many gets very hard on the computer, and therefore we cannot use the walls themselves as landmarks. That would make the program too heavy. However, I will save the findings of the walls, and plot them along with the landmarks (plotted with different marks), which will show the total findings of the robot, and illustrate why it is such a bad idea to use walls as the landmarks.

With landmarks placed, the maps might look like this:



Picture 3: Map 1 with landmarks



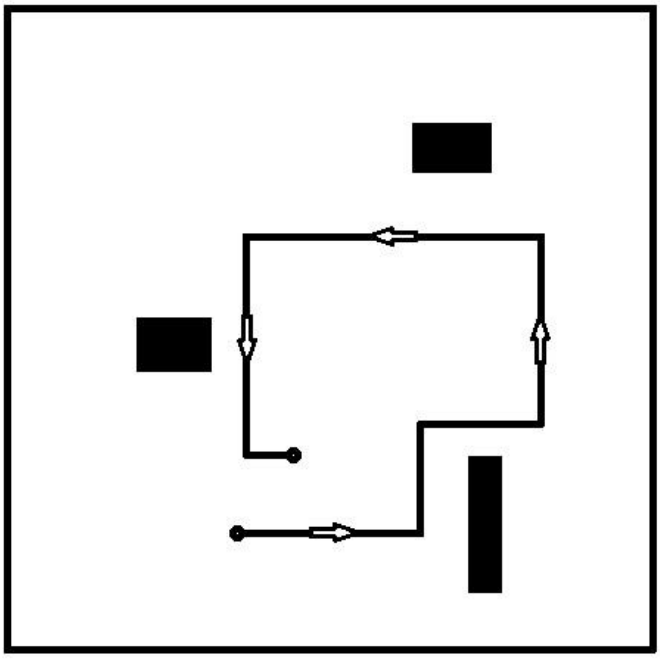
Picture 4: Map 2 with landmarks

In the pictures, the red circles are landmarks. As you can see the amount is quite low, in these pictures there are six of them, which makes for easier computing. Right now, I am using six landmarks, but obviously, one can use a few more, as long as it is in a moderate amount.

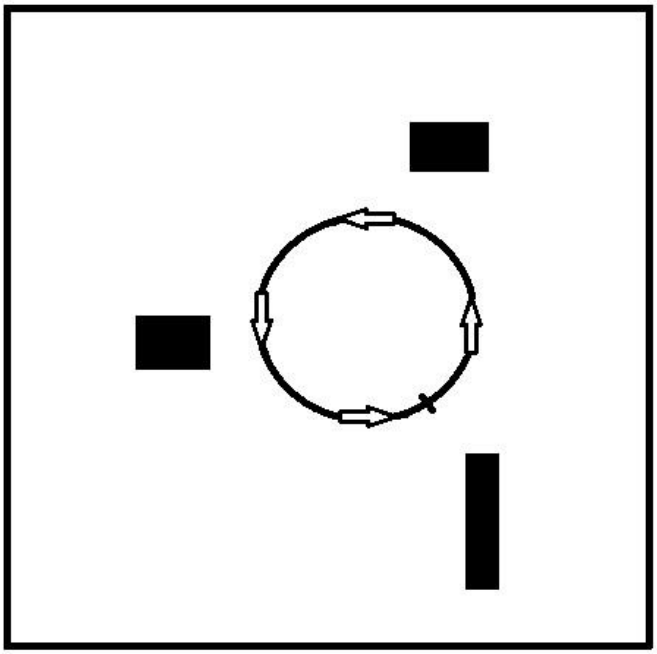
As mentioned I will describe in an upcoming section (the next section), that I will use premade routes in these maps. However, in the first map I will use 2 different routes, and therefore we have a total of three examples. Though, the landmark positions will remain the same as above.

3.2 Description of the Routes

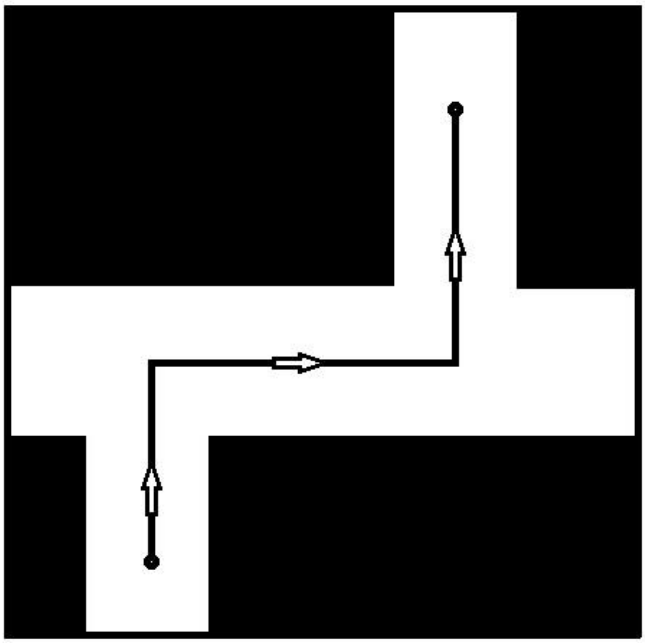
In this report I will use three different examples to illustrate the program. These examples are simply three different routes, whereas two of the routes are in the first map, and the third route in the second map. The following pictures show the three different routes within the maps.



Picture 5: Map 1, Route 1 (Track 1)



Picture 6: Map 1, Route 2 (Track 2)



Picture 7: Map 1, Route 3 (Track 3)

These three routes, or tracks, will be used throughout the report.

3.3 Description of the Robot

What's next is to implement an actual robot into the map. I will be using a rectangular robot with 2 wheels. The robot will have a few major physical variables: length and width, wheel radius, distance between wheels, and its position in x, y and the angle theta as its state. However, only its position are values that will change, the others are fixed.

As a measurement the robot will be mounted with a sonar, with the capability of scanning 360 degrees around the robot. The sonar will have two variables, its range and its interval size (in degrees).

3.4 Dynamic Model

As mentioned earlier, the robot needs a dynamic model, or motion model. Based on simple 2D movement, a rectangular robot with two controllable wheels will have a model like the ensuing:

$$x_{t+1} = x_t + T\frac{V_{R,t} + V_{L,t}}{2}\cos(\theta)$$

$$y_{t+1} = y_t + T\frac{V_{R,t} + V_{L,t}}{2}\sin(\theta)$$

$$\theta_{t+1} = \theta_t + T\frac{V_{R,t} - V_{L,t}}{l}$$

T is the sampling time, the interval between measures, $V_{R,t}$ and $V_{L,t}$ are the right and left wheels velocity, respectively, and l is the distance between the wheels.

3.5 SLAM Algorithm

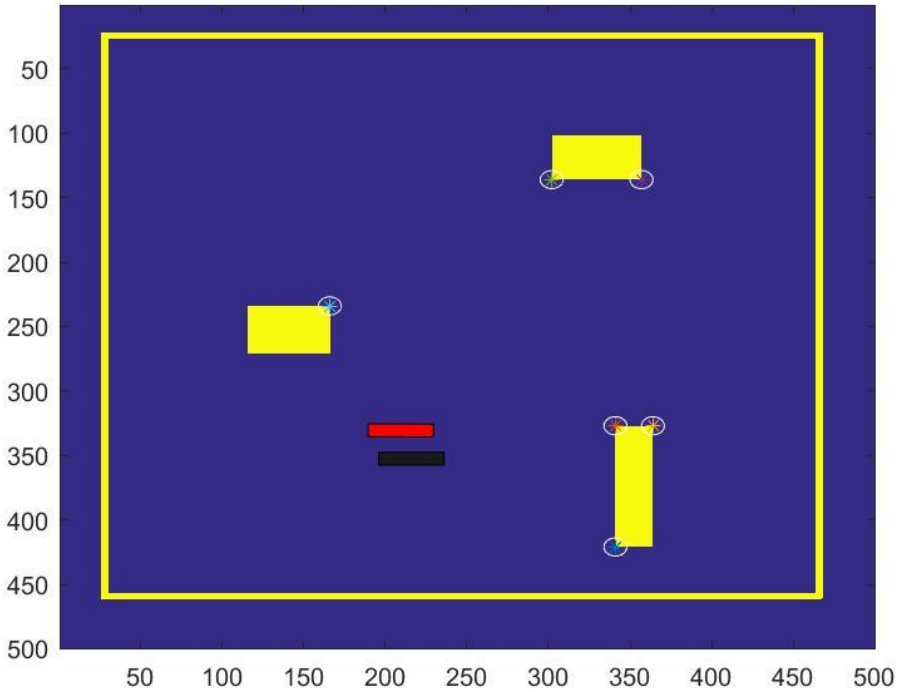
The algorithm will be run as a separate file, from the main file, and has a few inputs: the true robot pose, the current estimate of the position, its speed (used to calculate control vector), its change in angle, and two other variables used to find out which landmarks has already been seen.

It has three main parts: the prediction step, the correction step, and landmark initialization. As in the algorithm depicted previously, the prediction step will always be run. However, only one of the correction step or landmark initialization can be run at a time, based on if the landmark is newly or already found. In SLAM, the measurement are landmarks.
This is the main difference with the EKF.

Lastly, the algorithm will return values made within the program, and use them to plot new movement for the robot in the main file.

3.6 Illustration of Results

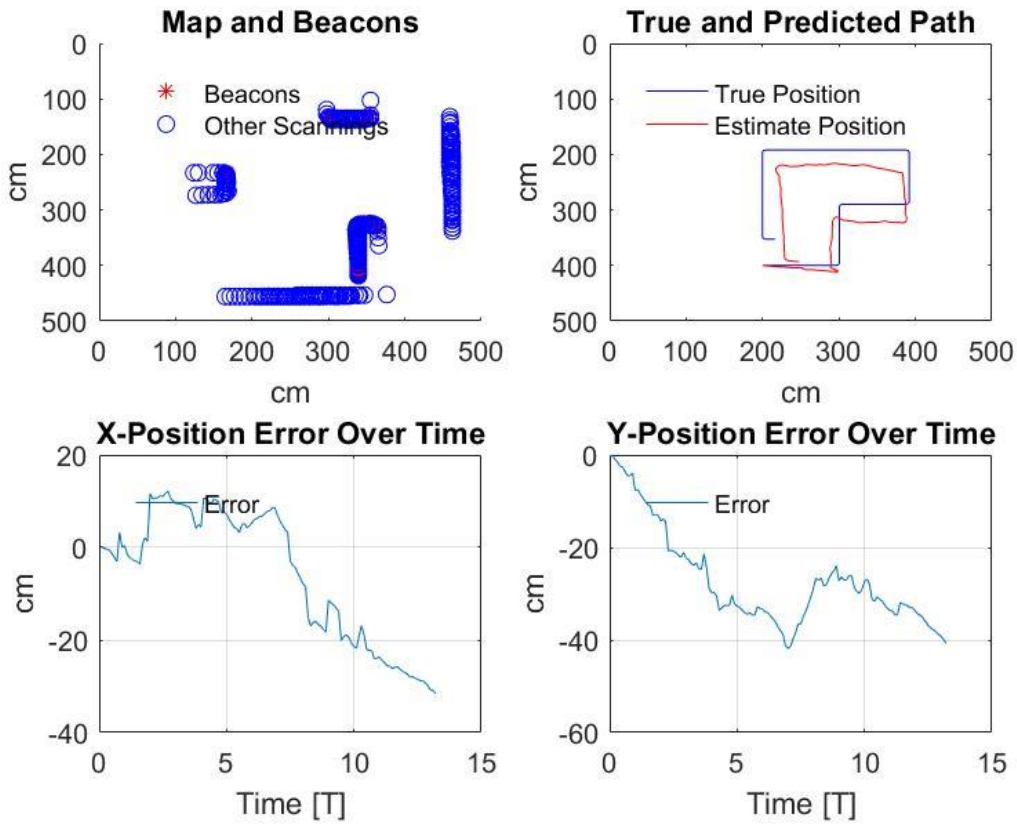
In order to illustrate the algorithms, I have implemented two robots into the program, one true (black robot) and one false (red robot). The true robot depicts the true movement, where the robot actually is. The false robot however, depicts where robot believes it is, based on the estimates given from the estimate. The following is a picture showing both the robots in the map:



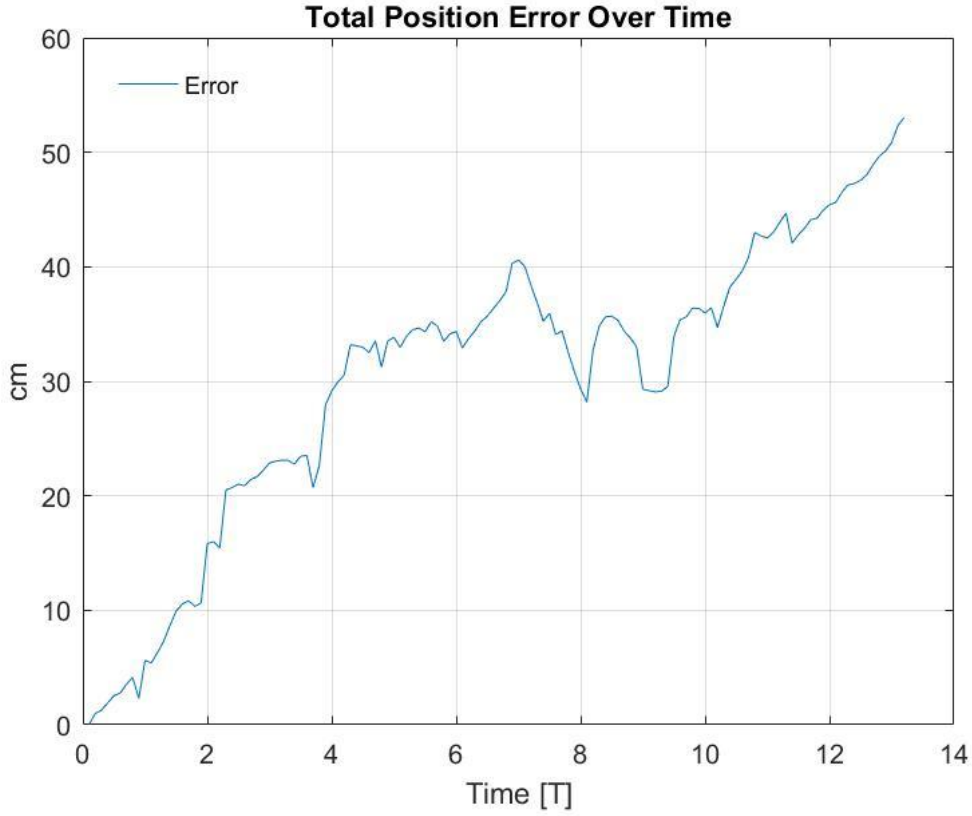
Picture 8: Map 1 with robots and beacons

Picture number 8 (above), shows how the map can look. The blue color is the background (previous white), and all the yellow objects are walls the robot can't hit (previous black). Colors are now introduced to separate between the two different robots. On some of the walls, one sees how landmarks looks (the circles with stars within them).

When the program finishes, 5 graphs will display. For example, it might look like this:



Picture 9: Illustration of results, 4 graphs in one



Picture 10: Illustration of results, last graph

The four first graphs, in picture number 9, gives us a lot of valuable information. The first quadrant shows a map of all walls the robot found, and the landmarks on them. However, it is hard to see in this picture, due to the scale making everything look small. For a bigger version, run the belonging program.

The second quadrant shows us two paths. The blue path of the total true position, which we don't know. The red path shows us the total estimated path, the path of the false robot, where we think we are. This is a great way to see how close we were to our true path.

The third quadrant and fourth quadrants shows us the error, in both x- and y-position, versus time. This is interesting, however, one clearly wishes to see the total error, which is what the fifth graph shows us. Here, one sees how every now and then the error decreases. This is due to the correction step. More on this will be explained in the upcoming chapter, Results and Discussion.

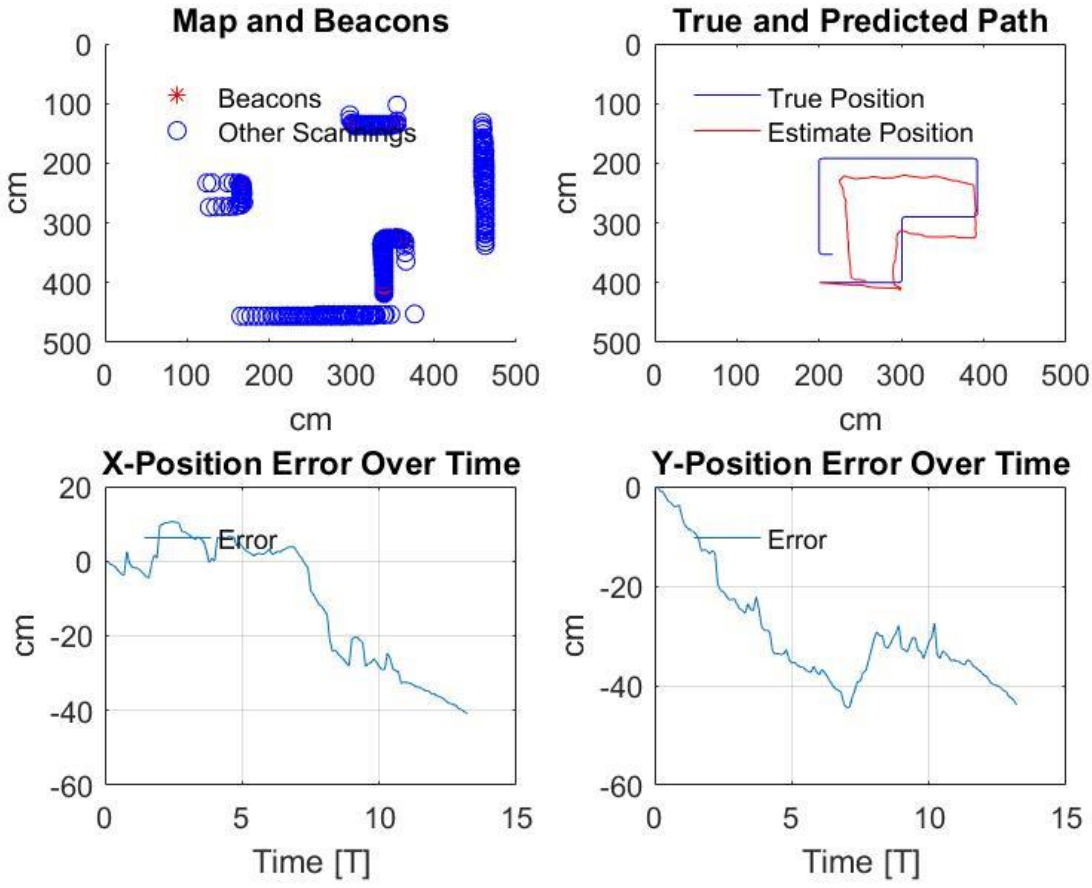
As mentioned earlier, it would be unwise to use the walls as landmarks for this task. In the first quadrant at picture 9, one clearly sees how many findings of the walls there are, which would cause for enormous matrices within the SLAM algorithm. My computer would crash be trying to handle that.

4. Results and Discussion

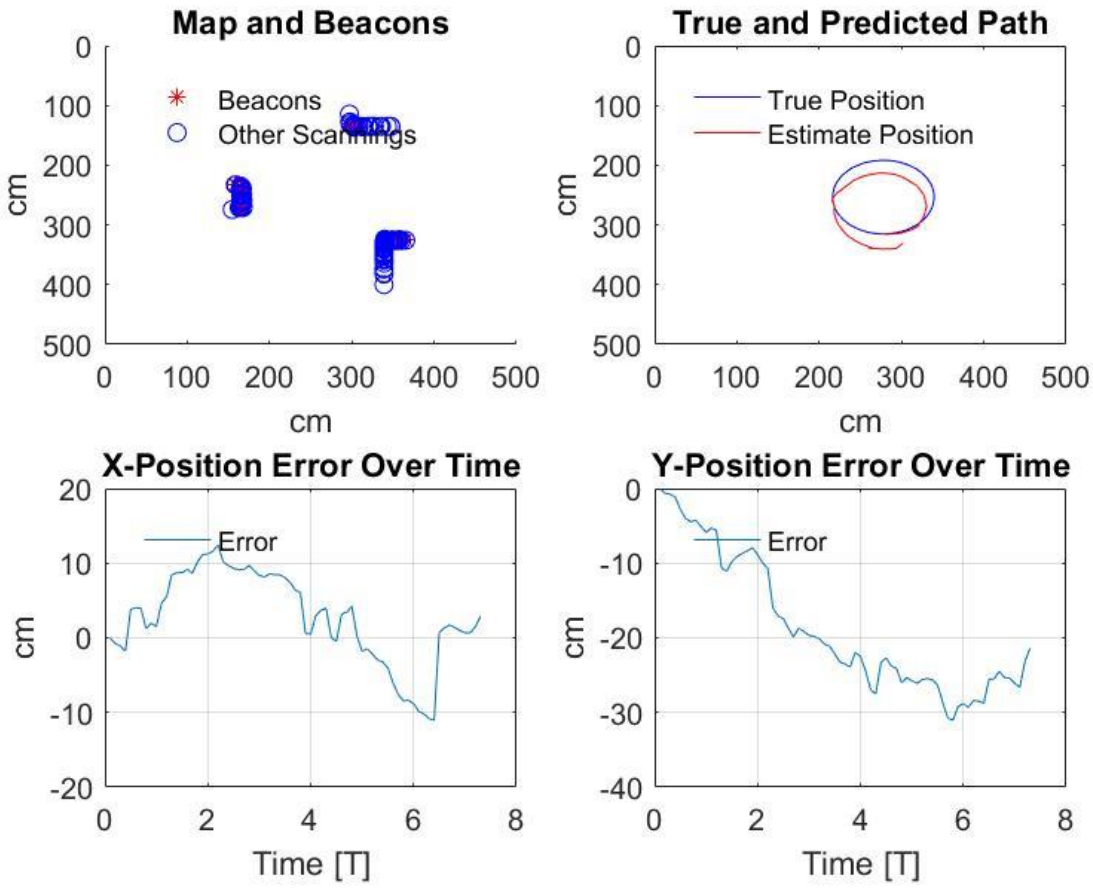
To make sure the program and algorithm works, good illustration is essential. This chapter will through pictures show the results of the program in a few different examples, as well as include a discussion of the results at the end.

4.1 Algorithms used in different Examples

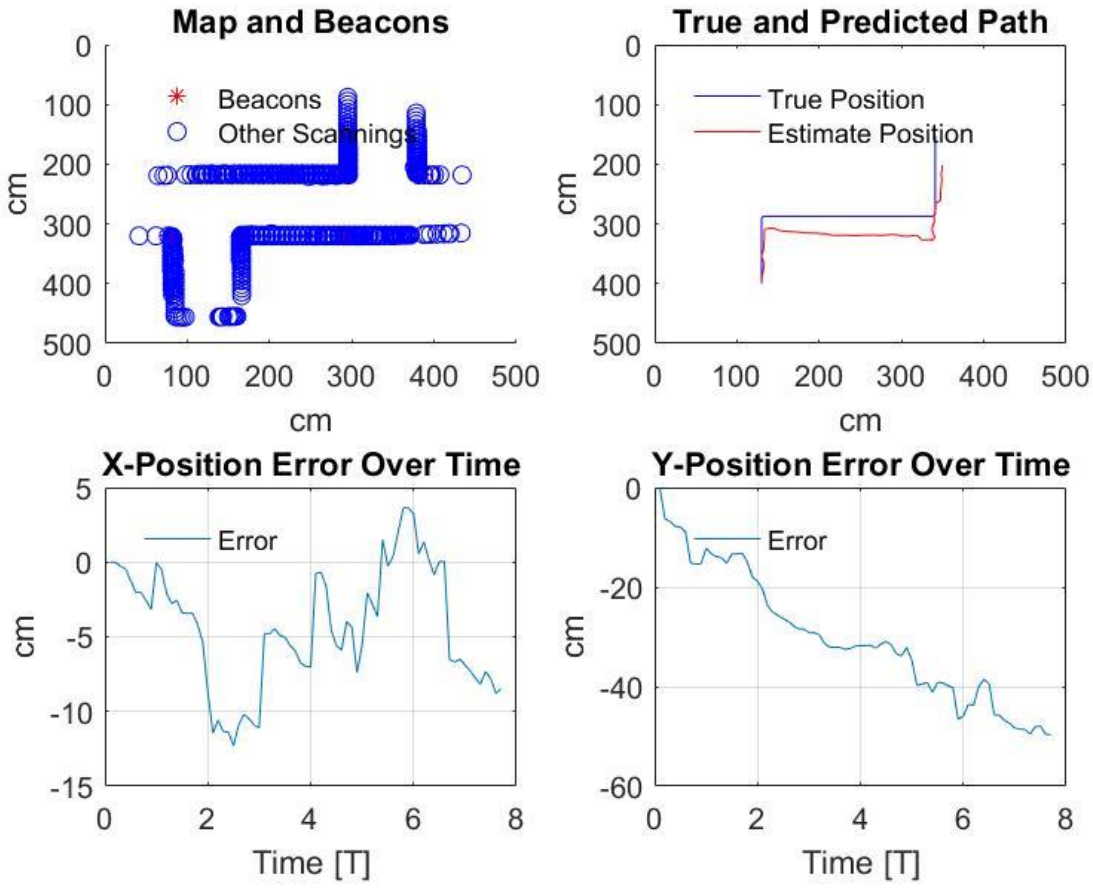
For this report I used two different maps, one with two different tracks within it, and the other one with only one track. For these three cases I used 6 landmarks, but obviously it is possible to use either more or less. Each case is shown by 4 separate pictures showing mapping, entire true and estimated path, x position error and y position error. The results were as follows:



Picture 10: Map 1, Track 1

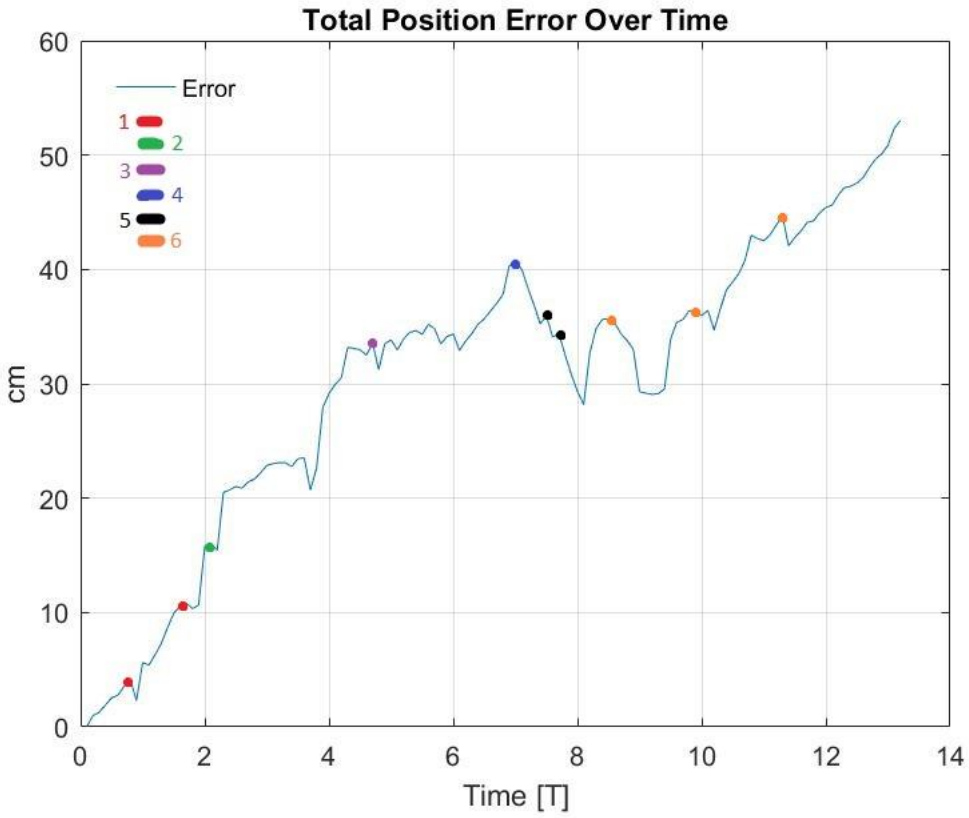


Picture 11: Map 1, Track 2

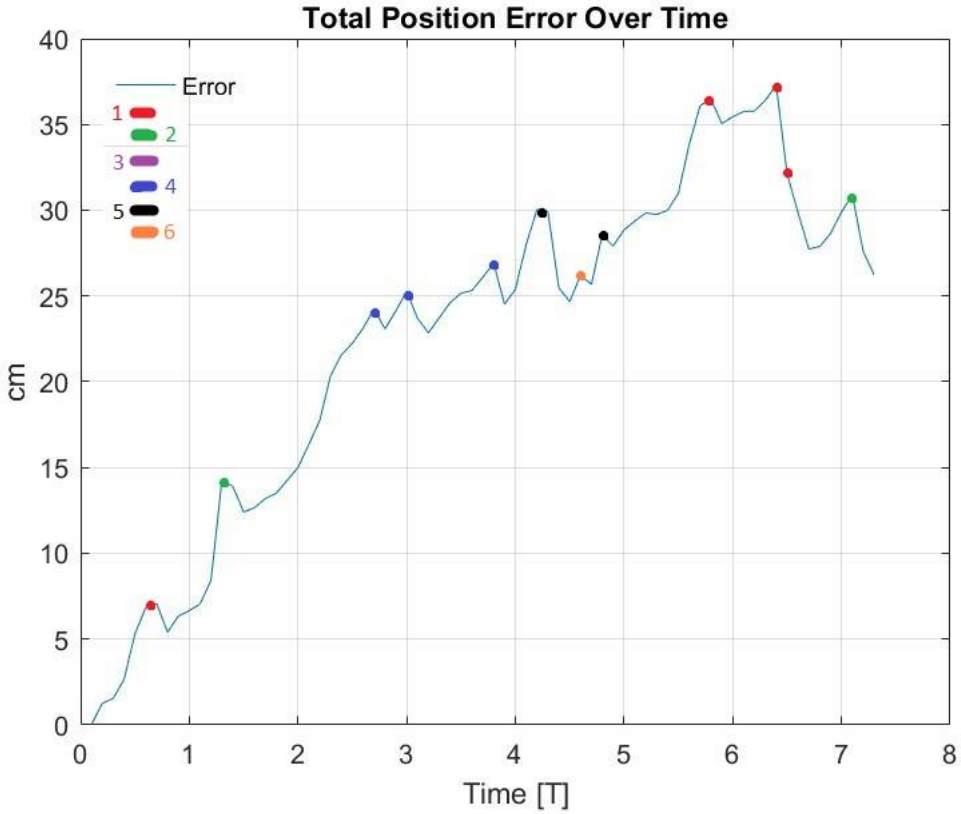


Picture 12: Map 2, Track 1

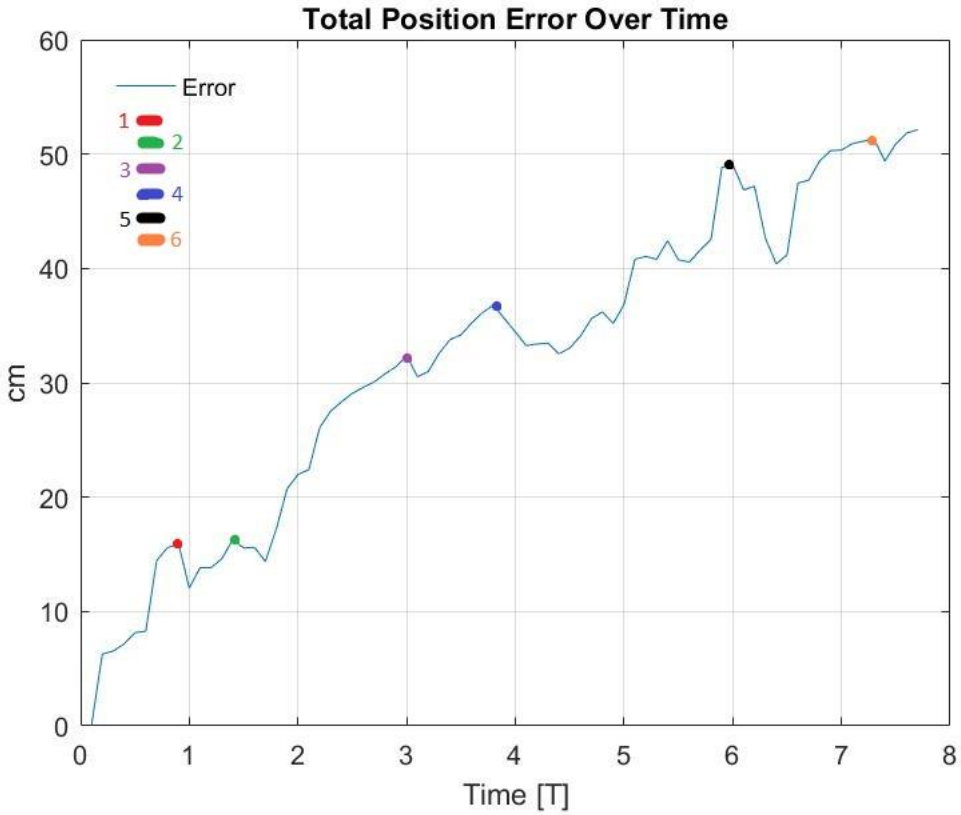
It is easier to see how the correction step corrects its position by looking at the total distance error, rather than the x-error and the y-error split up. One here sees how the error always grows but it now and then corrected and therefore lowered. Precisely as it should, as it is only corrected once it finds a beacon. The following pictures are the same, but marked whenever a beacon is found for the second time, and thus activates the correction step.



Picture 13: Total Error, Map 1, Track 1, Marked



Picture 14: Total Error, Map 1, Track 2, Marked



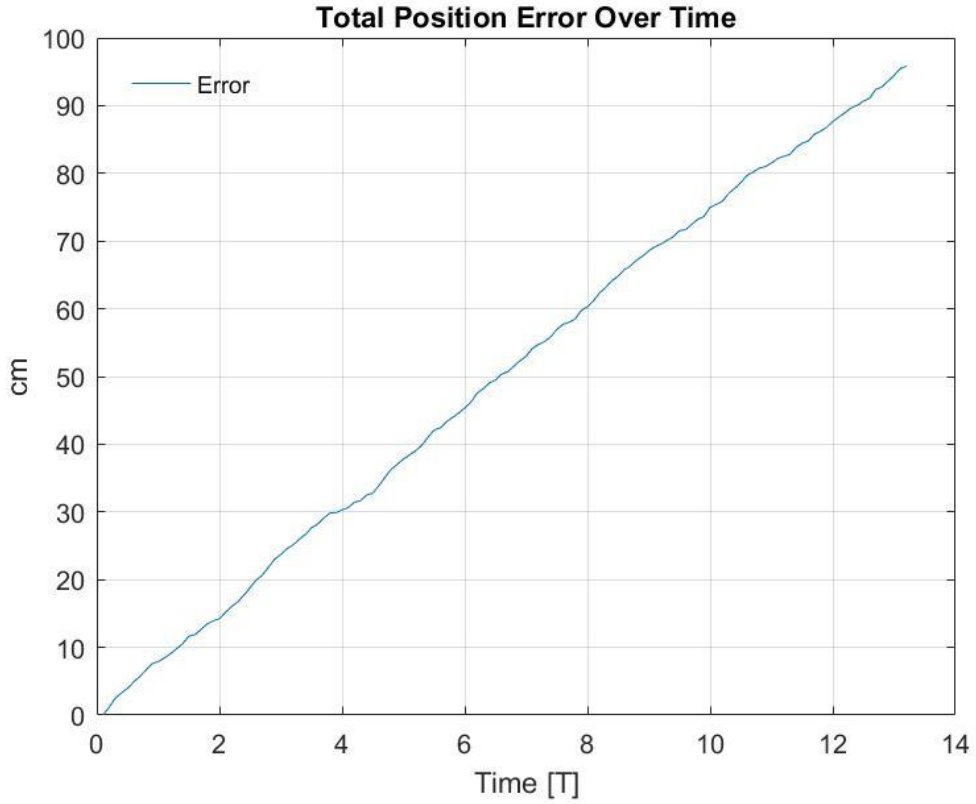
Picture 15: Total Error, Map 2, Track 1, Marked

4.2 Discussion

The three above pictures, marked whenever an already mapped beacon is found, shows how the correction step of the SLAM algorithm comes into play. However, is it hard to specify exactly where the landmarks are found. The scanning's are very frequent small in angle, which can make the robot find the same beacons over and over again within a few time steps. That is why we sometimes have a significant decrease in error after a beacon is found for the second time. Other times, the robot only finds an already mapped beacon once, and therefore the error is not reduced very much. The exact time stamps of where beacons (both old and new) are found, is displayed whenever the MatLab program is ran. Take a look at picture 14, which is of track 2, map 1 (a circle track). This causes the robot to see the first landmarks once again as it completes its track, which causes a big error decrease as the picture portrays.

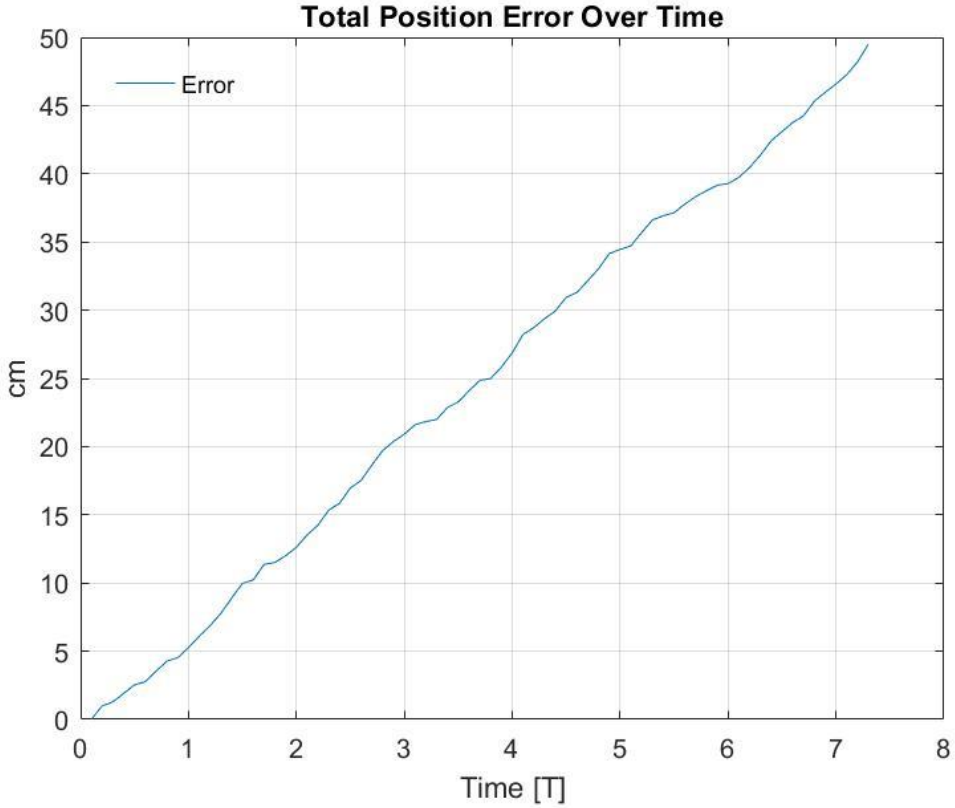
A good question is to ask why it is that sometimes after a beacon is found for the second time (in the marked graphs 16-19), the error continues to decrease a while afterwards. This is due to the fact that after it is found for the second time, it is often also found a third time, and a fourth, and so on. This means that the correction step will have the chance to be run several times, and thus decreasing the error significantly. However, this depends on the route, robot speed and sonar quality (faster and wider sonar range). Since a low robot speed and high sonar rotation speed causes the robot to see the landmark more times before it passes it, the correction step will be used more often. Same concept with route, one can plan the route to see already found landmarks over and over again. In reality one can accordingly argue that this makes the route planning, speed of sonar rotation and speed of robot an optimization problem of its own: How fast does the robot need to get to a place, without sacrificing too much position data?

If one where to run the program without any beacons, theory says only the prediction step would be active since we don't acquire any measurements, and consequently the error would only increase over time. The following picture shows the result of running with the first map and the first track:

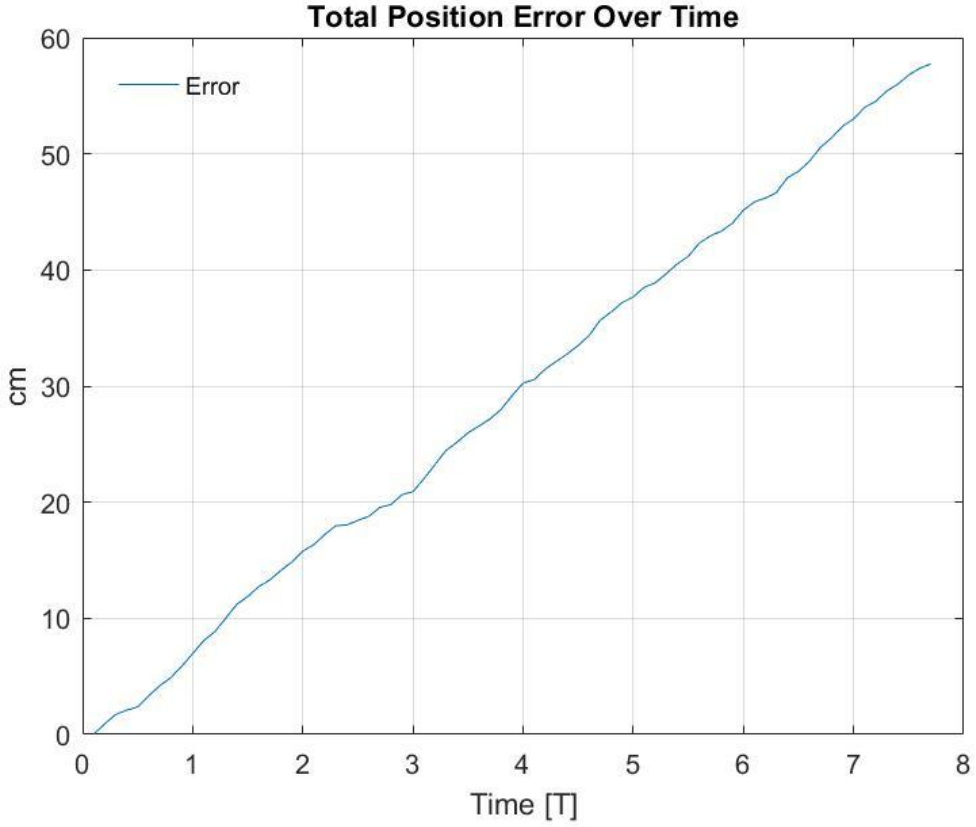


Picture 16: Total Error, Map 1, Track 1, No beacons

We see that the experiment goes hand in hand with theory. After tests, this is the case for the other two examples as well. The results of the two other cases are the two following pictures.

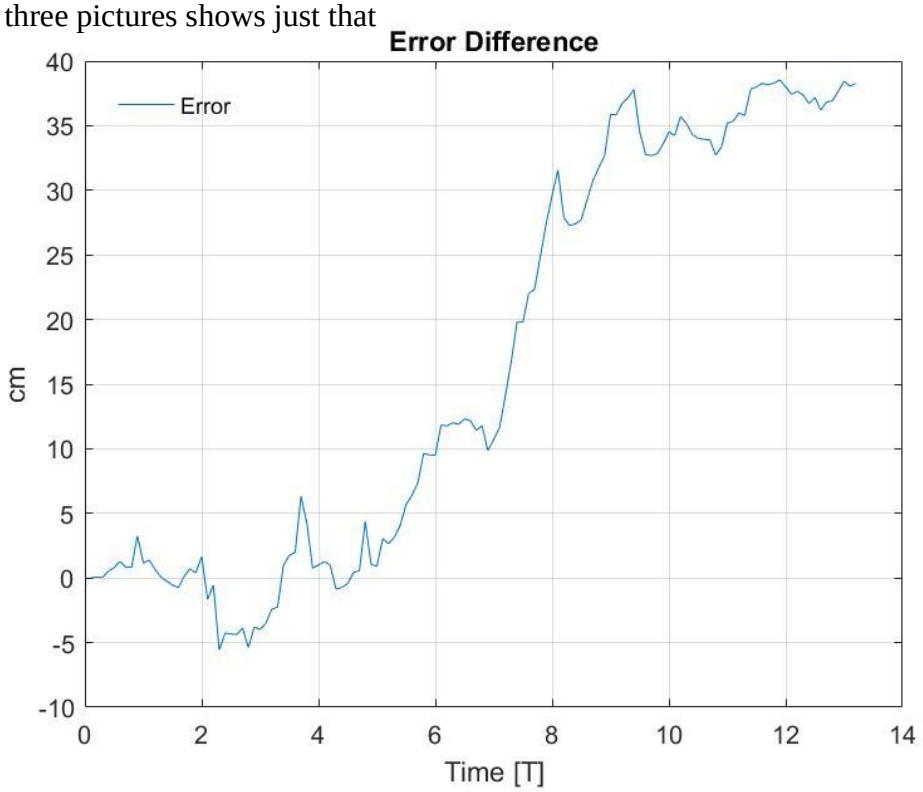


Picture 17: Total Error, Map 1, Track 2, No beacons

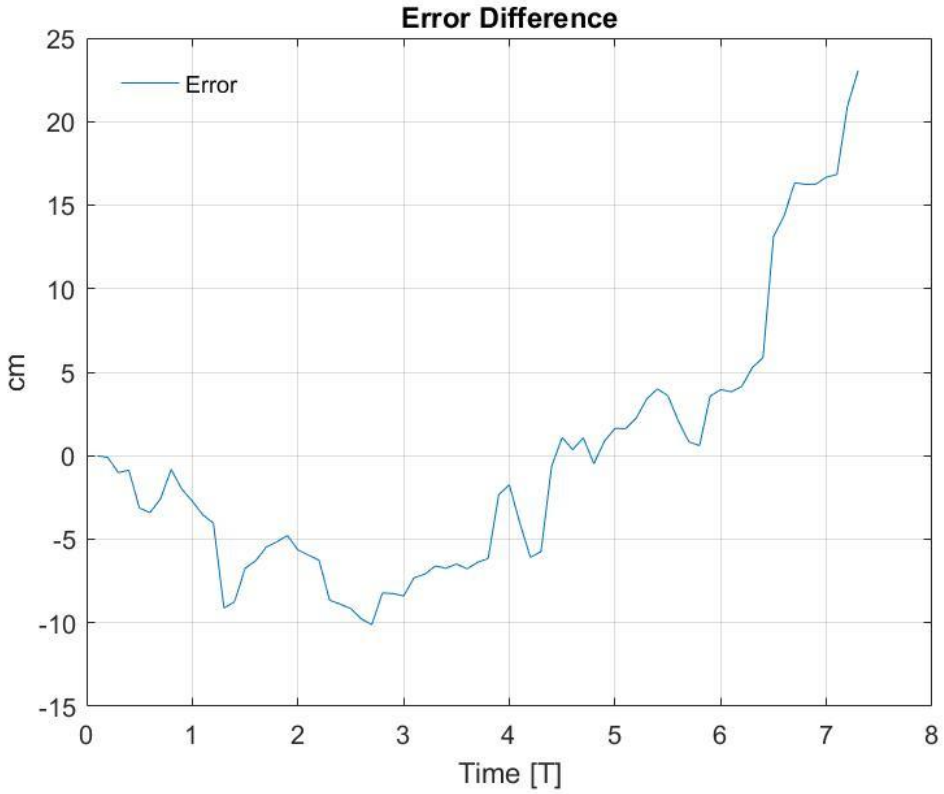


Picture 18: Total Error, Map 2, Track 1, No beacons

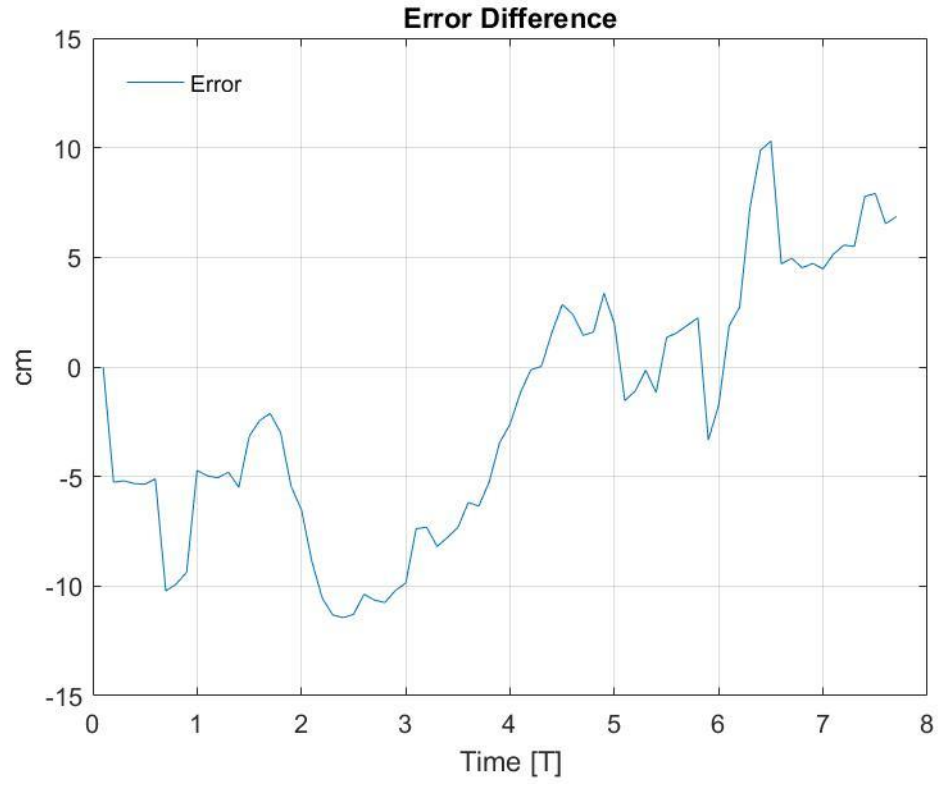
One would also want to compare these results to the ones with landmarks. By subtracting the error without landmarks and with, we can find out how much bigger it is at each point. The following three pictures shows just that



Picture 19: Map 1, Track 1, Error difference



Picture 20: Map 1, Track 2, Error Difference

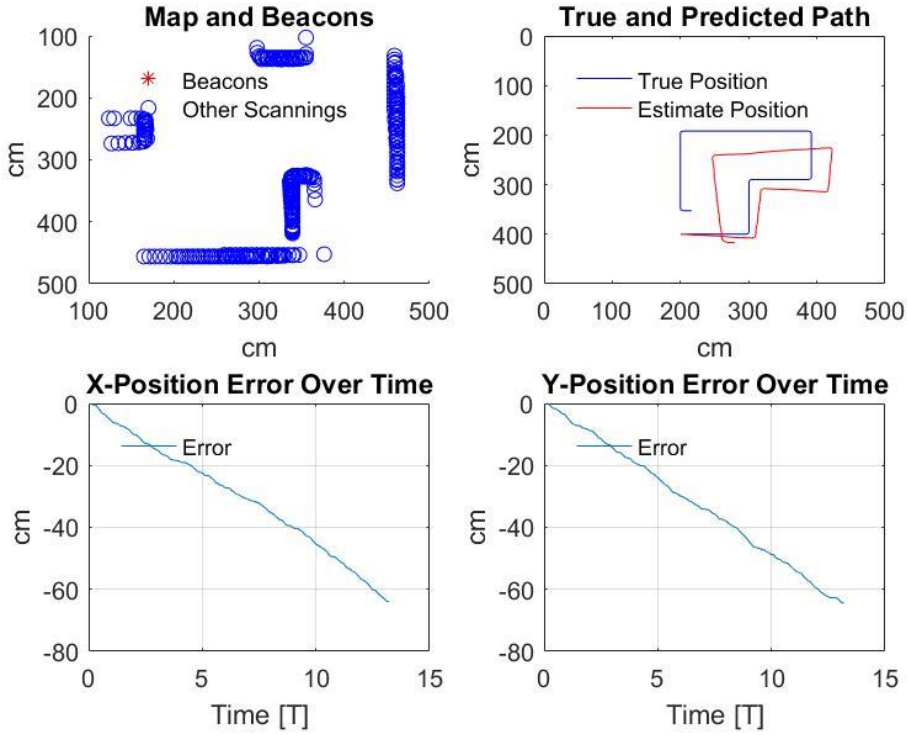


Picture 21: Map 2, Track 1, Error Difference

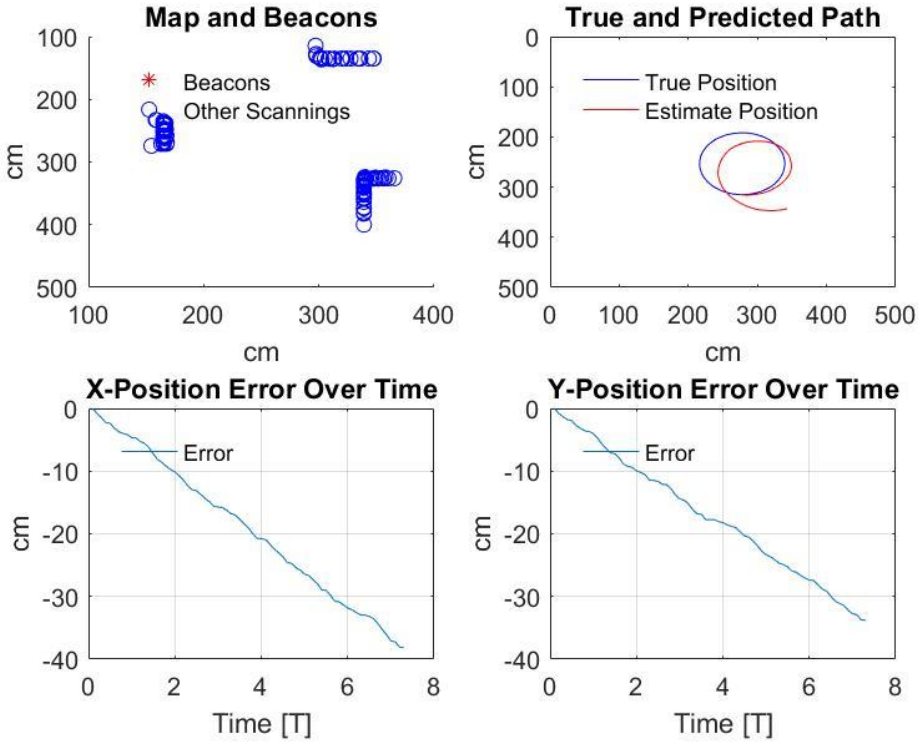
These above three graphs show the total error of the tracks without landmarks, subtracted by the total error of with landmarks. That means that one here sees that in the end, the error ends up

bigger a lot bigger without landmarks than with. Therefore, the correction step does work and helps the estimation.

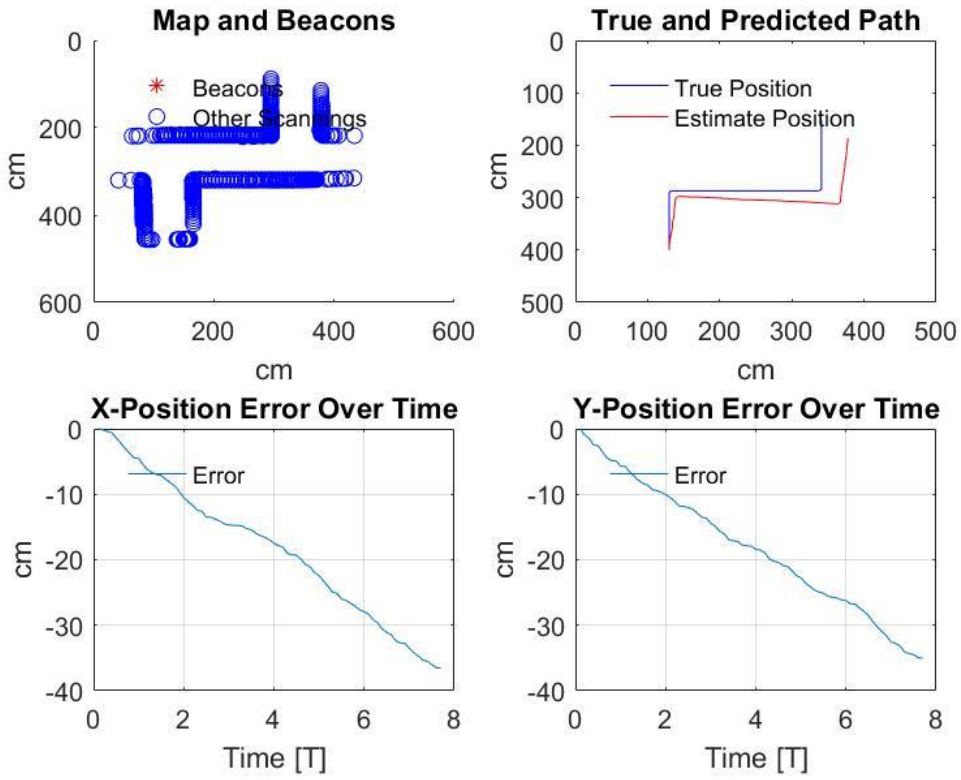
For more extensive information, it is also interesting to look at the total results (the 4 first grapsh) of the three cases, without beacons. They were as follows:



Picture 22: Map 1, Track 1, No beacons



Picture 23: Map 1, Track 2, No beacons

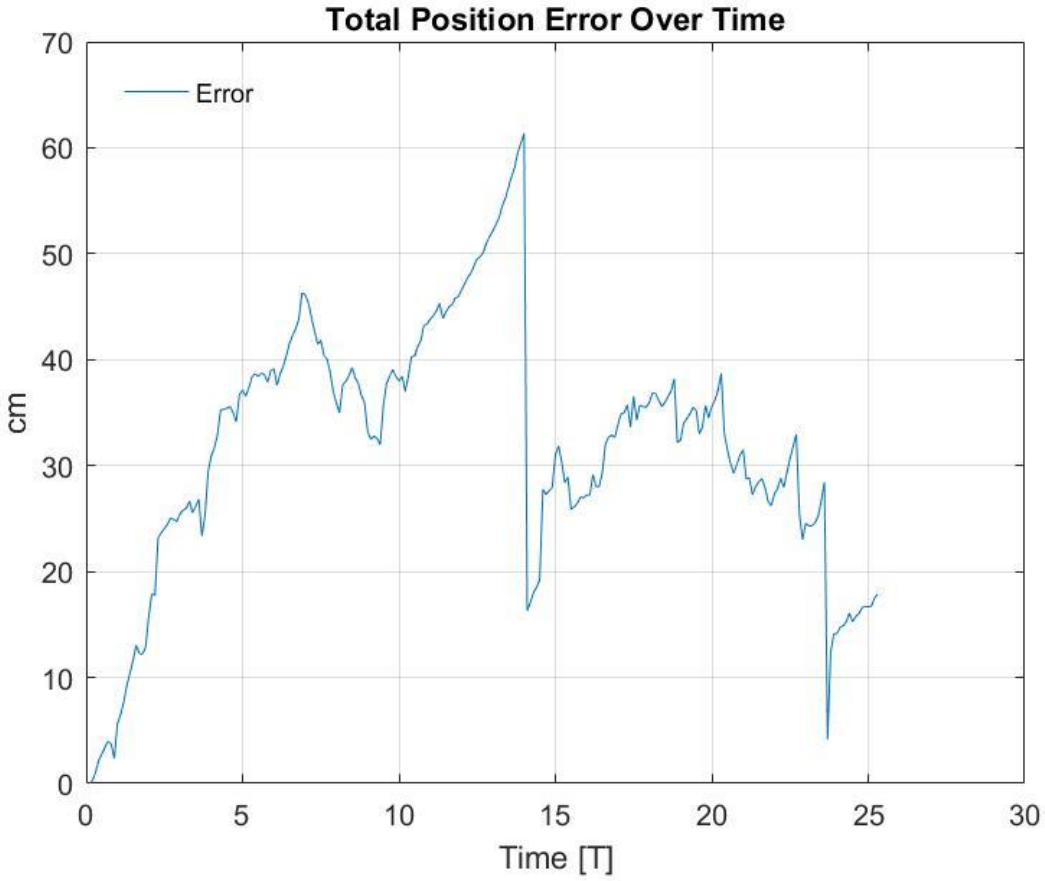


Picture 24: Map 2, Track 1, No beacons

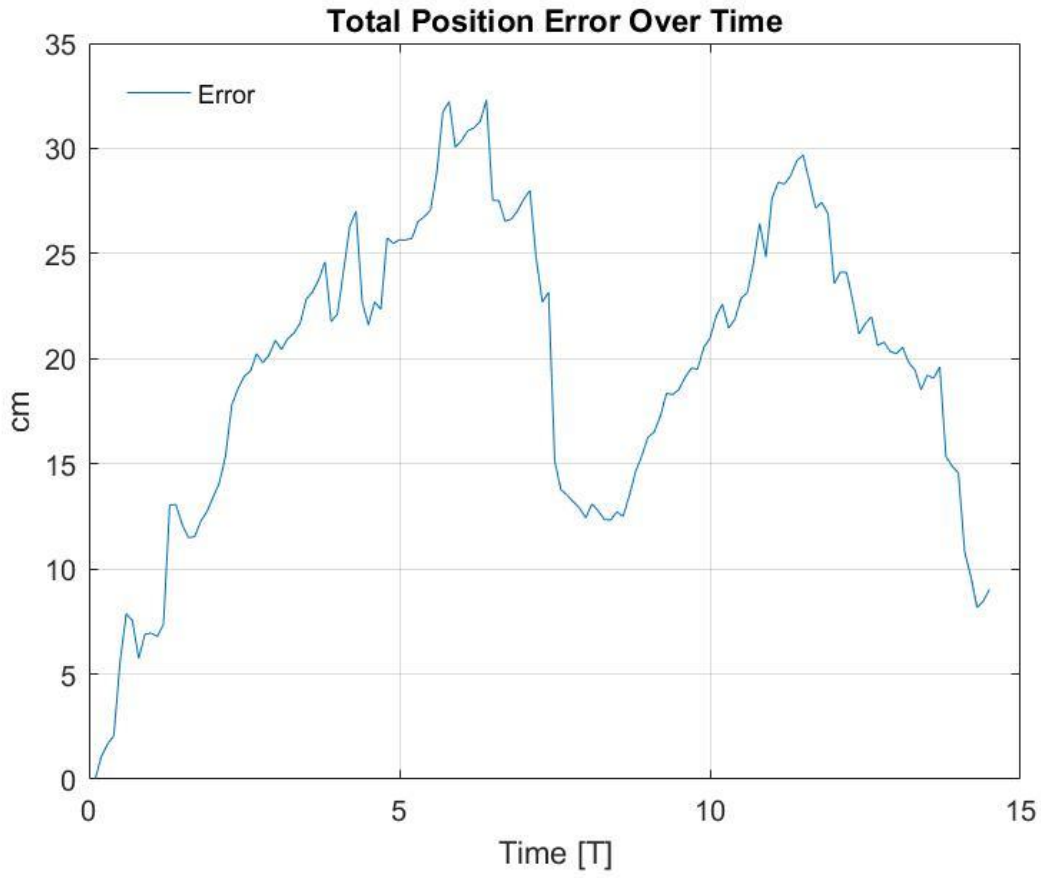
In these pictures, specifically the first quadrants, it is hard to see but no landmarks were found. Other than that, one also sees here in the second quadrants how the path of the false robot grows larger over time from the true path.

4.3 Doubled Path

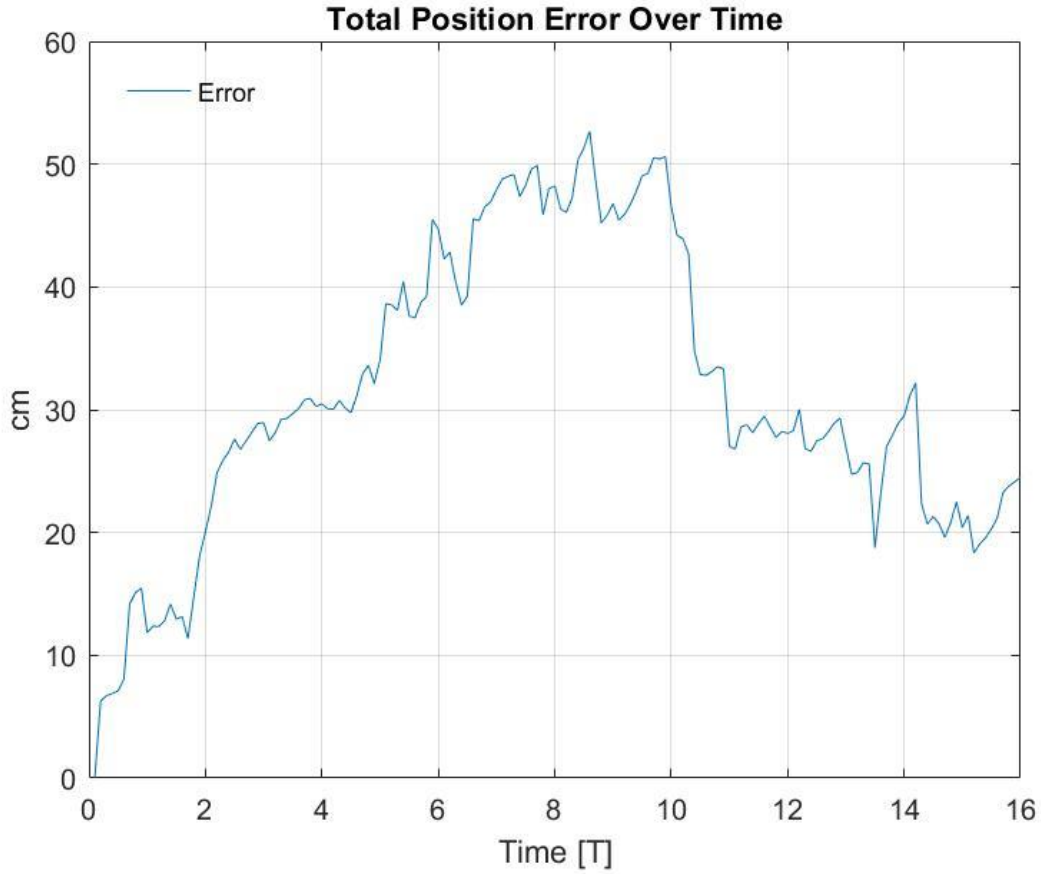
We know from theory that it is the correction step that reduces the error. Therefore, seeing previous landmarks more times should reduce the error a lot more. To show this effect I will here show the results of the three examples ran with the robot traveling its path twice. In the first two examples, the robot will continue its path and simply do it twice, meanwhile in the third example in the second map, the robot will make a U-turn and return to its original position. Following is the total error of the three different examples.



Picture 25: Map 1, Track 1, Double path



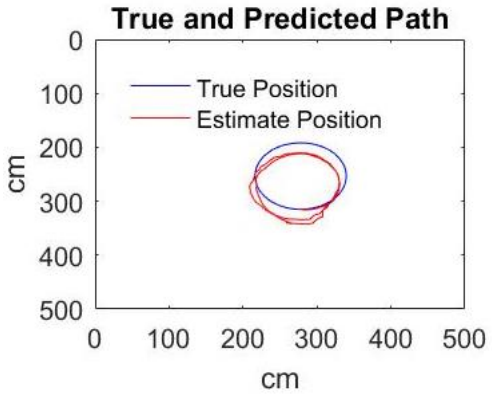
Picture 26: Map 1, Track 2, Double path



Picture 27: Map 2, Track 1, Double path

First let's explain picture number 28. As we can see at *T=14*, the error is significantly reduced when it sees the first landmarks once again. After that, the error is growing and being reduced further by continuing its path.

In the second track, picture number 29, we see that the error is greatly reduced and then rising, to be even more reduced further on. This has to do with the fact that the track is a circle. Take a look at the picture to the right. We see that that position error is very small once the paths of the false and the true robot are crossed, to then rise as it travels further from the crossing points. However, the error is still being reduced, as the false robot's circle is being more aligned to the true ones.



Finally, in the third example, picture number 30, we see that the error is being reduced here as the robot turns and sees all the landmarks over again. This is just as expected.

What we have witnessed now is a very good proof of the algorithms functionality, as with the logic used here, the error would be reduced even more by having the robot keep doing the same path.

5. Conclusion and Future Work

In this work we have discussed how a three different estimation algorithms work, the Kalman Filter, the Extended Kalman Filter and the SLAM algorithm, which are essential for robot localization. This work can be used as a beginner's guide to Estimation Theory and its implementation combined with the belonging MatLab files. For more in-depth information, one should perhaps consider looking at Probabilistic Robotics by Sebastian Thrun and Wolfram Burgard.

This project has led me to believe that SLAM is a great algorithm, with lots of future applications, and improvements to come. Although this project was only made in two dimensions, I am really interested to study how far we have come in the field of three-dimensional state estimation. We can also dare to have the robot ask more questions than "Where am I?". We could have it ask "what is around me, and how can I use it in my favor?". This however is a very complex task, and perhaps research is already being made on this topic.

My interest in improving SLAM has led to multiple ideas of future work, that would be reasonable to continue with after this project. For example, what would happen if we had moving and accelerating beacons? Or perhaps we could have multiple robots sharing the data collected and thus localizing and mapping faster in a collected map? It would be very interesting to look over these subjects, and hopefully both I and the reader of this thesis will be inclined to do so.

6. Bibliography

1. Probabilistic Robotics, Sebastian Thrun of Stanford University CA, Wolfram Burgard of University of Freiburg Germany, Dieter Fox of University of Washington WA. Published year 2000.

2. SLAM course, Simultaneous localization and mapping with the extended Kalman filter, Joan Solà. Published October 5, 2014. Available at http://www.joansola.eu/JoanSola/eng/course.html

3. Correction Robot pose for SLAM based on Extended Kalman Filter in a Roush Surface Environment, Jaeyong Park of Yeungnam University Korea, Sukgyu Lee of Yeungnam University Korea, Joohyun Park of Yeungnam University Korea. Available at http://cdn.intechopen.com/pdfs-wm/6313.pdf

4. Differential Drive Steering + Kalman Filter Basics, ejkreinar. Available at https://cwrucutter.wordpress.com/author/ejkreinar/

5. DESIGN OF AUGMENTED EXTENDED KALMAN FILTER FOR REAL TIME SIMULATION OF MOBILE ROBOTS USING SIMULINK, Iraj Hassanzadeh of University of Tabriz Iran, Mehdi Abedinpour Fallah of University of Tabriz Iran. Available at http://islab.ulsan.ac.kr/files/announcement/298/seminar.pdf

6. Extended Kalman filter – Wikipedia. Available at https://en.wikipedia.org/wiki/Extended_Kalman_filter

7. Kalman filter – Wikipedia. Available at https://en.wikipedia.org/wiki/Kalman_filter

8. Simultaneous localization and mapping – Wikipedia. Available at https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

9. Autonomous robots – Wikipedia. Available at https://en.wikipedia.org/wiki/Autonomous_robot

10. How a Kalman filter works, in pictures. Available at http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

11. SLAM for Dummies, A tutorial Approach to Simultaneous Localization and Mapping, Søren Riisgaard and Morten Rufus Blas of Massachusets Institute of Technology. Available at http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

12. "Big Dog" acquired by google. Article posted 14th of December by John Biggs. Article about this and the robot found at: http://techcrunch.com/2013/12/14/google-buys-boston-dynamics-creator-of-big-dog/

13. Bilgin's Blog, Kalman Filter for Dummies. Written March 2009 by Bilgin Esme. Available at http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies

14. How to write a thesis – School pf Physics. Written 1996 by Joe Wolfe (modified 2/11/06) and available at http://newt.phys.unsw.edu.au/~jw/thesis.html

15. Developing a thesis – Harvard Writing Center. Written 1999 by Maxine Rodburg. Available at http://writingcenter.fas.harvard.edu/pages/developing-thesis