UKF SLAM WITH SONAR SENSOR

-PROJECT IN APPLIED ESTIMATION-

Rasmus Göransson
Kungliga Tekniska Högskolan
Stockholm, Sverige

2012-01-09

ABSTRACT

This is a rapport for the individual project for the course Applied Estimation. I chose to build a Lego Mindstorms NXT robot and record data instead of doing a simulation to create data. The robot has tracks and sonar sensor. Data from the servo motors and sonar sensor is recorded with a custom built matlab program using Mindstorms NXT Toolbox from http://www.mindstorms.rwth-aachen.de. The data is then processed as a problem instances for Simultaneous Localization and Mapping (SLAM). The algorithm is based on the Unscented Kalman Filter and the development is divided into several steps such as recording data, calibrating sensors, localization without landmarks, localization with landmarks, and finally UKF SLAM.

INTRODUCTION

This is a report for a project in a course in Applied Estimation. I thought it would be interesting to apply what I have learned during the course on a real problem and not a simulation, so I wanted to build a robot and an environment and do Simultaneous Localization and Mapping (SLAM).

Building a robot was made easier by using Lego. Lego Mindstorms NXT is basically a toy for children and adults and makes it easy to build and program simple robots. The sensors and servos have poor precision but are easy to use.

THE ROBOT

The robot has three motors and one sonar sensor. Two of the motors are used for driving the robot and the third one is used to rotate the robots head. The sonar sensor is attached to the robots head. Since the head can be turned without turning the whole robot, measurements can be made in different directions without worry for slipping or loosing track of the robots heading. The robot also has a light sensor attached to its head but it is not used in this project. I named the robot "Infinity" because of the shape of the wires. See figure 1.

The robots pose is modeled as a 2D position with one heading. The turning of the robots head can be incorporated in the measurement model directly instead of having it in the pose.

Odometry from the three motors are recorded. The motors have built-in encoders which keep track of how many degrees the motors have turned since the robot was started. The encoders are fairly accurate for this purpose but the transmission and slippage of the tracks introduces a lot of error.

The measurements of the environment are taken from the sonar sensor. It is operated in 'snapshot mode' which means that it can record up to eight echoes for each pulse. This means that several landmarks can be seen at the same time.

Figure 1: The robot "Infinity". (The wires form the shape of the infinity symbol ∞).

THE ENVIRONMENT

At first I wanted to build a maze with walls and openings to be better prepared for the Robotics and Automation course. But because we have looked at using point based features in the course, environments were built using the legs of chairs as landmarks. This makes the problem easier because the same landmarks can be detected from different positions and modeled as points in the plane.

Figure 2 shows the first environment consisting of two chairs. A recording was made while the robot was driving around in an unstructured manner around and under the chairs. This dataset is referred to as dataset 1.

RECORDING DATA

I built a graphical matlab program to control the robot and record data. The program uses Mindstorms NXT Toolbox from http://www.mindstorms.rwth-aachen.de in order to communicate with the robot.

The typical way of using the program is to switch on the robot, connect, choose a filename for the recording, start to record, control the robot, stop recording, save to disk, disconnect the robot, and turn off the robot. The data can also be previewed using a naïve filter which simply sums up the values obtained from the motion model as if the information was perfect. A snapshot of the program is shown in figure 3.

Figure 2: The first environment consists of the legs of two chairs. Dataset 1 was recorded while the robot was driving around in an unstructured manner around and under the chairs.

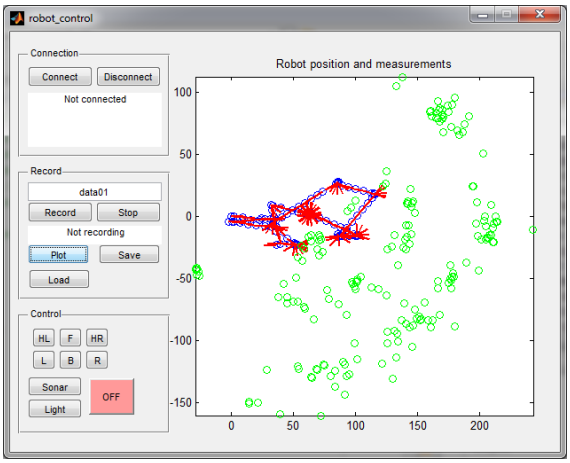

Figure 3: My custom built program for controlling the robot and recording data.

CALIBRATION

Odometry information is recorded from the two driving motors. The motors each keep a counter for the distance traveled and this distance is stored for each frame. These values do not directly tell how the robot is moving though. A model for the movement needs to be created and calibrated. The difference between consecutive readings should correspond to the velocity for

each motor. The sum of the two motors velocity corresponds to the robots velocity, the difference between the two motors corresponds to the angular velocity of the robot, but the values are not scaled properly by default. I created a simple motion model as seen in figure 4.

```
%% Extract velocity and angular velocity V, W
A = [data.entries.A]; % Head
B = [data.entries.B]; % Right motor
C = [data.entries.C]; % Left motor
echos = [data.entries.echos]; % Sonar sensor data
dB = diff(B);
dC = diff(C);
dF = dB+dC;
dH = dB-dC;
V = Fscale * dF; % Velocity in cm/timestep
W = Hscale * dH; % Angular velocity in radians/timestep
```

Figure 4: The motion model. `Fscale` and `Hscale` needs to be calibrated.

In order to calibrate the `Fscale` and `Hscale` coefficients, dataset 2 and 3 were recorded. Dataset 2 was created to calibrate the robots heading. The robot drives in a straight line, turns 180°, drives straight, turns again, this time accidently more than 180°, drives straight and stops. The first accurate 180° turn was what was needed to give a good estimate for the turning coefficient. The straight lines make the data easier to visualize. I started out with a guess and adjusted `Hscale` by hand until the angles looked right. The parameter was calibrated within about $\pm 5\%$.

The third dataset was created to calibrate the velocity of the robot and the sonar sensor. The robot was placed on the floor next to a leg of a chair and slowly driven backwards. The distance traveled was measured to 69.5 cm and it was easy to find a coefficient for velocity with less than 1% error. Sonar sensor data was also recorded in order to calibrate the sonar sensor. The sonar sensor is already calibrated to give readings in cm but the offset from the center of the robot was unknown.

The robot was rotated in place repeated times in order to determine the center of rotation. The robot could be seen to rotate around the front axis, which makes sense since the front of the robot is heavier. The distance to the leg of the chair from the center of rotation was measured to 10.0 cm. This makes it possible to compare each sonar sensor reading to a position along the robot path, determine offset and get an impression of the precision of the sensor. The measurements are shown in figure 5.

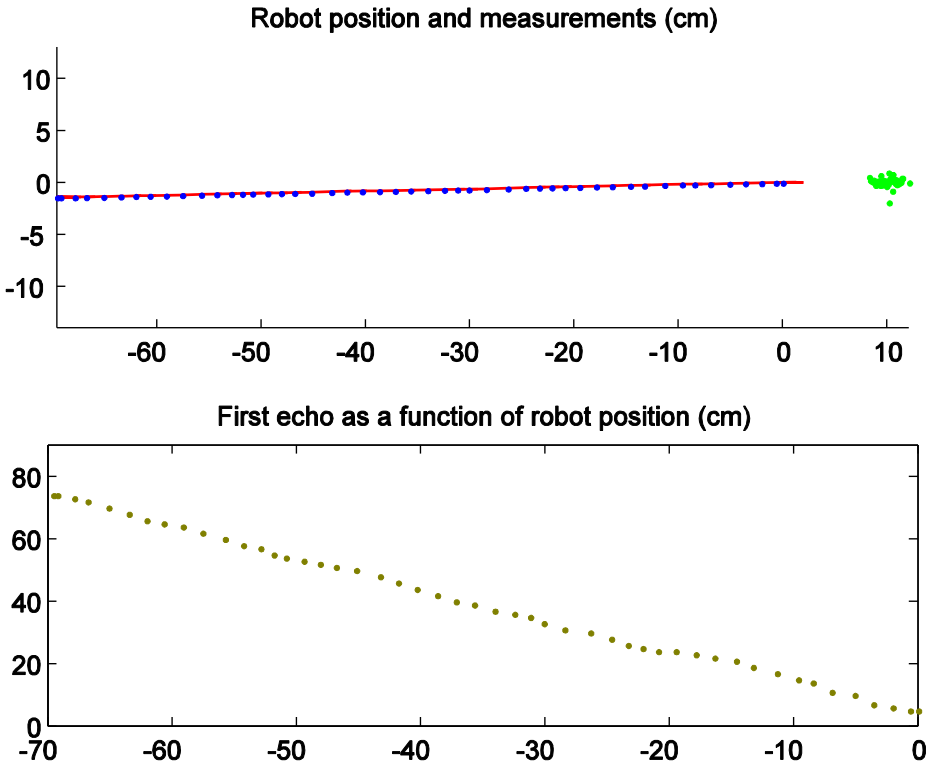

Figure 5: Sonar sensor calibration, the first plot shows naïve motion estimation along with estimated origin of the first echo for each robot position. Robot positions are plotted in blue, headings in red and measurements in green. The second plot shows the first raw echo as a function of the robots x-position.

UKF LOCALIZATION

I chose to use the Unscented Kalman Filter as the core of my algorithm. The UKF has several important benefits. According to S. Thrun [4], it should be at least as accurate as an Extended Kalman Filter (EKF) and may even have better accuracy. The computational load is said to be about the same or slightly higher. Another benefit is that no jacobians need to be calculated for the UKF.

The course put a lot of focus on the EKF but the UKF seems like a better alternative to me.

UKF replaces the linearization in EKF with the unscented transformation. Instead of computing a jacobian around the estimated mean, several sigma points representing the distribution are transformed through the motion model and analyzed on the other end to give a new Gaussian estimation of the posterior distribution. This reminds a bit of particle filters but avoids problems with stochastic sampling and still uses a Gaussian for estimating the state.

The selection scheme of sigma points described by S. Thrun seemed confused with many different variables and strange expressions spread out on several pages and partially hidden in the text. What's with all the $n + \lambda$ when λ subtracts n? I read on Wikipedia [3] that normal values are $\alpha = 10^{-3},\ \kappa = 0,\ \beta = 2$. It looks so much better after inserting the definition of λ, removing κ and simplifying. When I started to insert the equations into each other I was amazed by how much things could be simplified. Take a look at figure 6.

$\lambda = \alpha^2(n+\kappa) - n$	$\lambda = \alpha^2 n - n$ (not even used anymore)
$\gamma = \sqrt{n+\lambda}$	$\gamma = \alpha\sqrt{n}$
$w_m^{[0]} = \frac{\lambda}{n+\lambda}$	$w_m^{[0]} = 1 - \frac{1}{\alpha^2}$
$w_c^{[0]} = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)$	$w_c^{[0]} = 1 - \frac{1}{\alpha^2} + (1 - \alpha^2 + \beta)$
$w_m^{[i]} = w_c^{[i]} = \frac{1}{2(n+\lambda)}$ for $i = 1, \dots, 2n$	$w_m^{[i]} = w_c^{[i]} = \frac{1}{2n\alpha^2}$ for $i = 1, \dots, 2n$
$\mathrm{X}^{[0]} = \mu$	$\mathrm{X}^{[0]} = \mu$
$\mathrm{X}^{[i]} = \mu + \left(\sqrt{(n+\lambda)\Sigma}\right)_i$ for $i = 1, \dots, n$	$\mathrm{X}^{[i]} = \mu + \left(\alpha\sqrt{n\Sigma}\right)_i$ for $i = 1, \dots, n$
$\mathrm{X}^{[i]} = \mu - \left(\sqrt{(n+\lambda)\Sigma}\right)_i$ for $i = n+1, \dots, 2n$	$\mathrm{X}^{[i]} = \mu - \left(\alpha\sqrt{n\Sigma}\right)_i$ for $i = n+1, \dots, 2n$

Figure 6: Simplification of UKF parameters

Why wasn't this done in the first place? Looking at the equations now, most of it seems to make sense. It is now visible that the center weights add up to 1. But what is $\sqrt{n}$ doing in γ? I decided to find out more and as I dug deeper I found an article describing several selection schemes [1]. This made me realize that there is more than one way to do it. Following references from that article lead me to some articles from the Robotics Research Group, University of Oxford, describing the unscented transform and the parameters [2]. The choice of parameters mainly affects how derivatives of higher order are captured.

After some testing with different values on α, without seeing much difference on the results, I decided to set $\alpha = \frac{1}{\sqrt{n}}$ to bring each sigma point exactly one standard deviation out from the mean. The reasoning behind this is that they should capture the nonlinearities of the distribution at some distance from the mean.

In the equation for the sigma points, there is a square root of a matrix. There are several ways to define and find roots of matrices, according to [2], any square root will do. Matlab have several ways of calculating square roots. After testing with the principal square root `sqrtm` and the Cholesky factorization `chol`, I found no difference between the results and decided to use `chol` since it was recommended on Wikipedia [3] as a numerically efficient and stable method.

There is also (at least) two ways of handling the process and measurement noise, either add it after the transformation and drawing new sigma points from the resulting distribution, or augmenting the state with the noise parameters and only draw one set of sigma points. According to S. Thrun, the second approach is to be preferred. I decided to follow that recommendation and did not try the first method.

LOCALIZATION WITHOUT LANDMARKS

The first development step towards localization and SLAM was to do localization without correcting against landmarks. This means using only the prediction step and not the update step. The naïve filter was used as a reference to compare with. I implemented the UKF prediction and calibrated the noise parameters until the results looked good. The process noise is controlled with four parameters controlling how much velocity and angular velocity is affecting the noise in velocity and angular velocity. I realized that slipping happens mostly on acceleration and deceleration but I the model do not take that into account. This should be something to investigate further. The noise model is shown in figure 7.

```
%% Control noise covariance M
% TODO: Add error by acceleration
R = zeros(2, 2, size(V,2));
R(1,1,:) = Ralpha(1) * V .^ 2 + Ralpha(2) * W .^ 2 + 1e-9;
R(2,2,:) = Ralpha(3) * V .^ 2 + Ralpha(4) * W .^ 2 + 1e-9;
```

Figure 7: The noise model with four noise parameters in `Ralpha` The extra `1e-9` is to avoid numerical problems when generating sigma points.

LOCALIZATION WITH ONE LANDMARK

The next development step was localization with one known landmark. By using only one landmark, the association problem is avoided. Dataset 3 is suitable for this since the closest measurement is always to the same landmark with known position. The certainty in x-direction is increased compared to localization without landmarks, see figure 8.

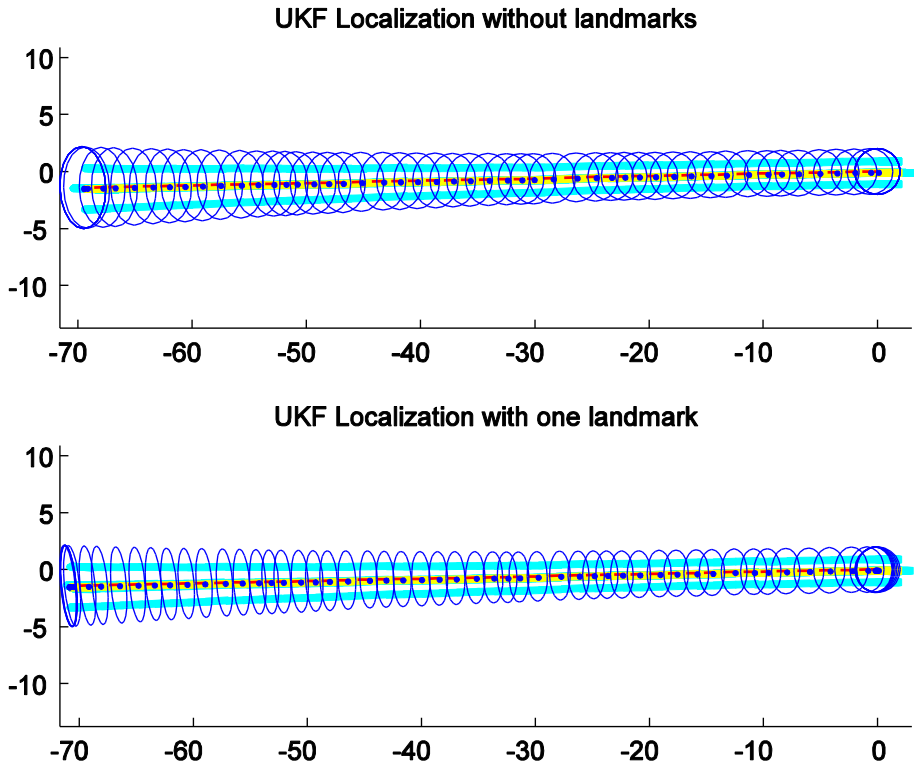

Figure 8: Localization on dataset 3 without landmarks compared to localization with one landmark. Notice the how the uncertainty in x-direction decreases significantly when a landmark is used.

LOCALIZATION WITH SEVERAL LANDMARKS

When there are several landmarks, the association problem arises. I solve the problem by choosing the most likely association separately for each echo from the sonar sensor with an additional test of the Mahalanobis distance to detect outliers. This approach may associate different echoes to the same landmark but since the echoes are already some distance apart, this risk is reduced.

An improvement would be to only consider landmarks within the field of view of the sonar sensor. This is not implemented yet though. Another improvement would be to see if there are several landmarks with high probability and creating separate hypotheses for each likely association. This is called Multiple Hypotheses Tracking (MHT) and would result in a Gaussian mixture posterior distribution.

The updates are done in a sequential manner, closest echo first. The closest echo has the most certain origin and has in general the least risk of getting wrongly assigned. The reduced uncertainty could improve the precision of the later assignments. This is not really correct since the same sigma points are used for every echo. This leads to exaggerated updates and will later be revealed as a critical source of error. This problem can be solved either by resampling new sigma points after each update or by postponing the updates and do them in a batch. I solved it by resampling new sigma points after each sequential update.

SIMULTANEOUS LOCALIZATION AND MAPPING

UKF SLAM is very similar to EKF SLAM. The state is extended with the positions of the landmarks in order to enable SLAM. This means that the dimension of the estimated state grows with two for each landmark. The number of sigma points grows with four for each landmark.

I decided to do divide the implementation in two steps. The first step is to do SLAM from a vague starting guess and refine the positions of the landmarks. The problem of creating new landmarks can thereby be postponed to the second implementation step.

SLAM FROM STARTING GUESS

Dataset 3 contains one chair with a known position of one leg. The positions of the other legs are not exactly known but a reasonable staring guess can be made. I took some rough measurements of the chair. The front legs are about 38cm apart, the rear legs are 34 cm apart the distance between the two pairs are about 38 cm. The chair is 10 cm away from the starting position and the seat is to the right of the robot. Using these measurements together with a covariance of 10 cm^2 gave a good result, see figure 9. Notice that my starting guess was wrong about the direction of the chair. The closest leg was a back leg and the chair was observed from the side. My starting guess describes the chair as observed from the front. The algorithm still manages to correct the positions of the legs.

I noticed that the algorithm failed if the uncertainty of the starting guess was 100 cm^2. The state covariance gets negative values in the diagonal, which should be impossible. This error was tracked down to the imperfect sequential update step, discussed in *Localization with several landmarks*. After fixing this issue, the problem disappeared. Figure 9 is actually plotted from the corrected version of the algorithm.

SLAM WITHOUT STARTING GUESS

The problem of creating new landmarks directly from sonar sensor measurements is that the sonar sensor does not estimate heading. The sonar sensor is sensitive to landmarks within a beam in front of it but only gives information about the distance. The heading to the landmark is not Gaussian distributed, neither is the location. The probability distribution of the location is shaped like a piece of a circle. Even if this is poorly described as a Gaussian, I decided to give it a try.

To determine the field of view of the sonar sensor, a new dataset was recorded from an environment with a single glass bottle on the floor. The robot starts out looking away from the bottle then slowly rotates bringing the bottle into view and keeps rotating to bring the bottle out of view again. The field of view can be observed by at looking the angles where the bottle is seen and where it is not. The angle was estimated to 45 degrees.

Representative samples can now be created as an arc at the appropriate distance from the robot. The mean and covariance of the landmark is calculated directly from the samples. When a measurement is detected as an outlier, instead of just disregarding it, a new landmark is created and appended to the state. The state grows dynamically for each new landmark. When new measurements are made of the same landmark the uncertainty decreases.

Figure 10 shows the result on dataset 3. Even though some landmarks are correctly identified, the lower landmarks fail to be identified correctly.

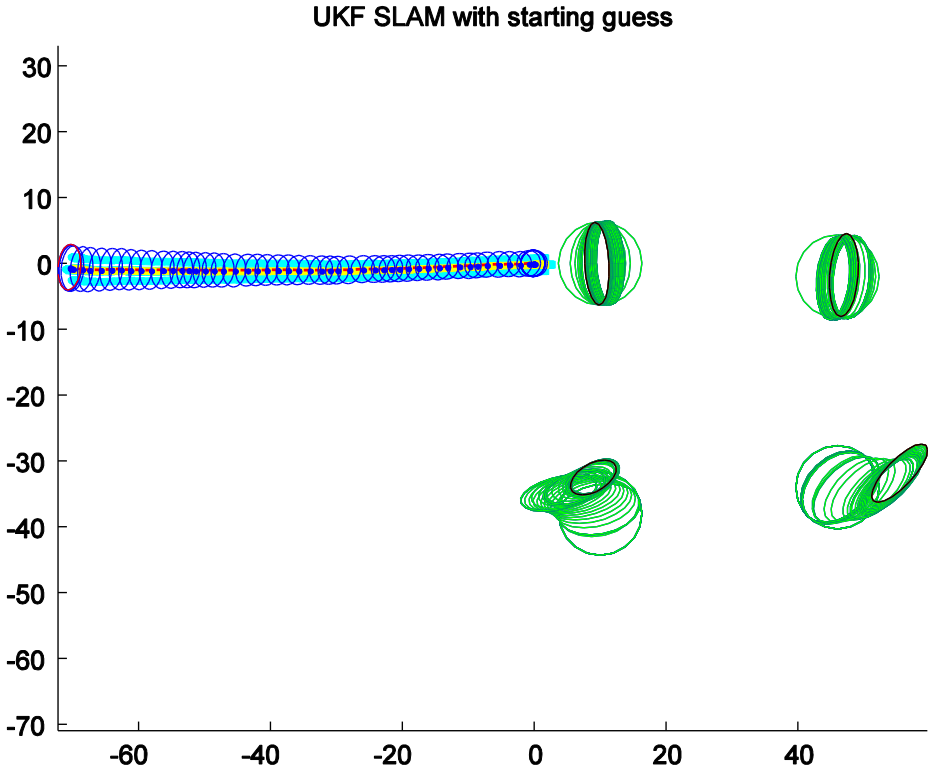

Figure 9: SLAM on dataset 3 from a starting guess. The starting guess describes a chair in an untrue direction but the algorithm still manages to correct the positions of the legs.

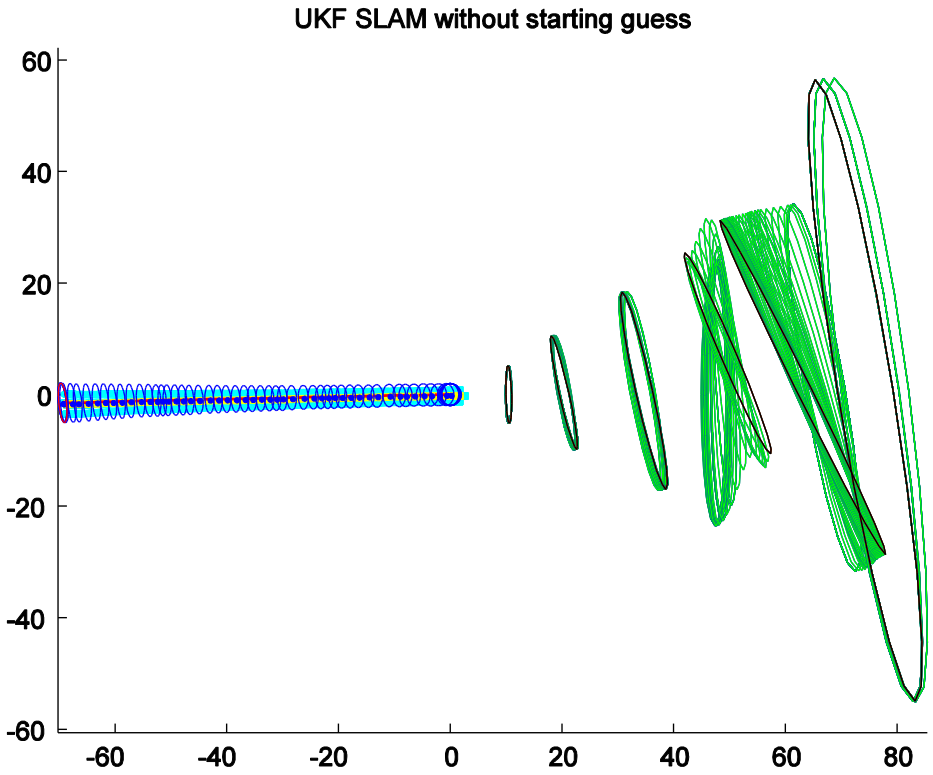

Figure 10: SLAM on dataset 3 without starting guess.

CONCLUSION

I have successfully built a robot and implemented localization and SLAM. The SLAM without starting guess gives quite poor results though. Even so I am satisfied with the results.

One thing I wanted to implement but did not have time to do was to create several hypotheses and keep track of them separately. A landmark which is not visible at first is generally introduced in one side of the robots field of view. By creating separate hypotheses for different angles of landmarks, a better estimate of the posterior distribution could be made. It might be enough to make two hypotheses for each new landmark, one to the left and one to the right.

It would be interesting to try out this implementation on bigger datasets and see how it behaves on loop closing.

The robots head was not used as it was intended. It was designed to be able to rotate and capture a bigger field of view. Driving around landmarks while looking at them could give good estimates of positions.

I feel like I have gained a better understanding of the concepts of localization and SLAM and of covariance matrices in general. The relation between samples and covariance and being able to transform between them without stochastic samples is something I really liked.

REFERENCES

[1] Richard J.B. Pieper. Comparing Estimation Algorithms for Camera Position and Orientation, Internship, Linköping Univeritetet, 2007

[2] S. J. Julier, J. K. Uhlmann and H. F. Durrant-Whyte. A New Approach for Filtering Nonlinear Systems. In The Proceedings of the American Control Conference, pages 1628–1632, Seattle WA, USA, 1995.

[3] Wikipedia http://en.wikipedia.org/wiki/UKF#Unscented_Kalman_filter, 2012-01-09

[4] S. Thrun, W. Burgard, D. Fox. Probabalistic Robotics. The MIT Press, 2006