TTK4250 EKF-SLAM

Ludvig Løken Sundøen, Nadia Garson Wangberg

August 12, 2021

1 Abstract

In this assignment Simultanious Localization and Mapping (SLAM) was implemented and tuned for a simulated and a real dataset. SLAM is the process of building a map of unknown surroundings while simultaneously locating an agent within it. EKF-SLAM was implemented, which estimated the position of a car while creating a map of the trees around it.

2 EKF-SLAM

EKF-SLAM estimates the state consisting of both the pose and map using an EKF. In this assignment the EKF process motion model used the wheel odometry as the input, while the measurement model used range-bearing measurements from a laser-scan. Through the JCBB algorithm the scan measurement are associated to landmarks in the map. The odometry is the integrated speed measurement from the wheel encoders. Spin and surface irregularities can cause an error in the speed measurement, which will accumulate, causing drift. Frequent aiding measurement in the form of landmark associations will correct this drift and ensure an accurate pose estimate.

2.1 Disadvantages of EKF-SLAM

The EKF linearizes its model at the estimated state $\hat{\eta}$. If the algorithm does not have a good initial state η_0 or if its estimate $\hat{\eta}$ is very far from the actual state, the linearisation could become inaccurate and the filter may quickly diverge. In addition the EKF assumes gaussian noise which sometimes is an oversimplification. Since the EKF lacks stability properties it is not as robust as other SLAM algorithms.

EKF-SLAM is a recursive-SLAM algorithm which only estimates the last pose, as opposed to full-SLAM where the entire pose trajectory is estimated. While we are usually only interested in the last pose, performing full-SLAM often takes advantage of the structure of the problem which could lead to better results. State of the art methods has largely converged to graph based methods solving the full-slam problem, like for example ORB-SLAM.

Another disadvantage with EKF-SLAM is its time complexity being $\mathcal{O}(M^2)$, where M is the number of landmarks. EKF-SLAM becomes very slow over time as more landmarks are associated. This is because the posterior covariance matrix P_k's correlated terms must be updated. P_k is initialised as a block diagonal sparse matrix, but as new landmarks are discovered the denseness grows. This is because landmarks are heavily correlated to each other and to the pose. This denseness is a central aspect of SLAM but also very concerning as for both space and time complexity.

2.2 The Inconsistency Problem

As seen in figure2 the EKF-SLAM performs really well on the simulated data. What is not shown here however is the inconsistency problem the EKF-SLAM suffers from. We added slight gaussian distributed noise to the odometry measurements, with $\mu = 0.0$ and $\sigma = 0.05$. The results are seen in plot1. Equivalent results happened when Q was tuned very low, putting a lot of trust the imperfect odometry.

In this scenario the EKF-SLAM was clearly inconsistent as it had several overconfident landmark estimates that were wrong. This is very noticeable in the top right landmarks, as their positions are clearly wrong but their covariances are very small and heavily underestimated. This means that too much trust has been put in the prior and too little in the range-bearing measurements. These inconsistent landmarks causes the trajectory to diverge, which is clearly seen when simulating for longer. The divergence causes the algorithm to run incredibly slowly, which is again because several additional unwanted landmarks was added.

The reason for this inconsistency is due to the EKF believing it has more information than it does in reality. It has an understanding that a shift in the xy-plane for both the robot and landmarks are unobservable. What it fails to understand however is that this unobservability also goes for a rotation of both the robot and the landmarks. This results in false information in the orientation direction, as can clearly be seen in fig1, where several false landmarks appears slightly rotated from the real landmark. This causes the trajectory to slowly diverge and the algorithm to run slowly. These results unsatisfactory and shows that plain-vanilla EKF is not a robust solution to the SLAM problem. Generally speaking, consistency of the EKF-SLAM can be improved through robocentric mapping, observability constrained Jacobians, Laplace approximation or using the XKF instead of the EKF.

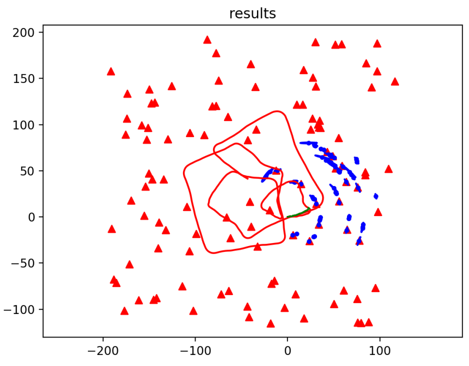

Figure 1: The inconsistency problem in EKF-SLAM

2.3 Tuning for the simulated dataset

Tuning the parameters for the simulated dataset is crucial for the algorithm to work with real data. If either the simulation data is not representative or the parameters are tuned too specifically, the tuning may not generalise to real world data.

The parameters were tuned according to several measures. NIS and NEES should be close to one to ensure consistency, while RMSE should be close to zero. We started by initialising Q and R to identity matrices. This resulted in both NIS and NEES being below one. Q was lowered until NEES was close to one. It seemed like the error in heading was smaller than in x and y coordinates. Therefore $Q33$ was tuned a lot lower than the other entries. This resulted in good NEES values as can be seen in figure 3. The R diagonal entries represents the variance in range and bearing respectively. R was tuned down until NIS approached one and most of the NIS was inside the 95% CI as can be seen in figure 4.

The other parameter were mostly set to the given values. The initial values of JCCBalphas given in EKFSLAM.py gave good results and were therefore not further tuned. The initial state were set to the ground truth value. The initial covariance P could therefore be set to zero. In the real world this is not usually feasible, since the ground truth is usually not available. Tuning these values was therefore done differently for the real data set. The final results for the average NEES and RMSE values can be seen in table 1 and figure 5. The trajectory is seen in figure 2.

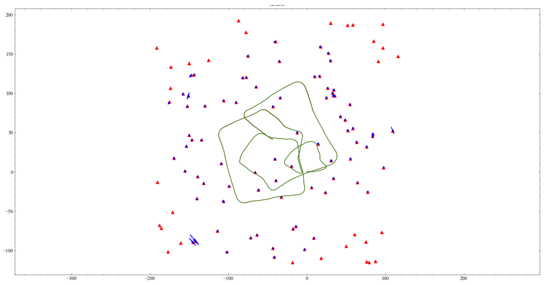

Figure 2: trajectory and map for simulated data

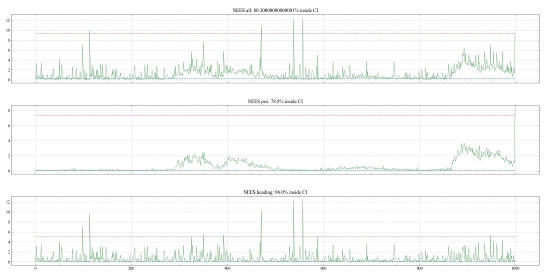

Figure 3: NEES in simulated data

2.4 Tuning parameters on VP-data set

Right away we noticed that the VP-dataset was a lot tougher than the simulated dataset. It included false alarms and missed detections for the landmark measurements. It was also longer with more landmarks which made the scenario more complex and run time became very slow. The EKF also assumes gaussian noise which may not be the case in real life. These are some of the reasons the results are worse than for the simulated dataset, as can be seen by comparing figure 2 and figure 7.

The parameters were initialised to those found in the simulated dataset. As the runtime was poor, tuning was done such that new landmarks were reduced. If the JCBBalphas were set too low too many measurements

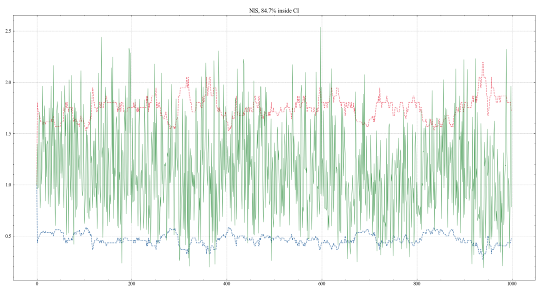

Figure 4: NIS with simulated data

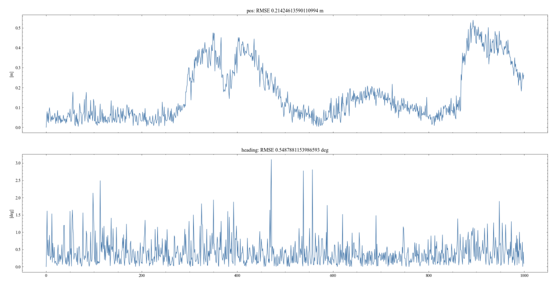

Figure 5: RMSE with simulated data

CI ANEES all	[2.8501,3.1537
ANEES all	1.1974
CI ANEES pos	[1.8779 2.1258]
ANEES pos	0.5892
CI ANEES heading	[0.9143 1.0895]
ANEES heading	0.6359
RMSE pos	0.2142
RMSE heading	0.5478

Table 1: Stats of simulated data

would be gated, which could result in slower runtime. Both tuning R and JCBBalphas higher could result in fewer landmarks, which could make this process run faster. Therefore JCCBalphas was set to $JCCBalphas_{real} = 5JCCBalphas_{sim}$. The resulting accuracy measures can be seen in 6 and 7.

A too small R makes JCBB unable to associate measurements with the stored landmarks, which leads to additional landmarks. On the other hand, A too large R makes the measurement a possible fit or possible association to any of the landmarks, which leads to a lot of wrong associations. With this in mind, R was set $R_{real} = 2R_{sim}$ which gave better results. This lead to both faster run times and a higher number of NISes inside 95% CI and $ANIS = 0.9871$.

The plots contain several errors. In figure 7 it can be noticed that the estimated path and the GPS coordinates differ. There are several reasons for this. First of all, GPS measurements are not perfectly accurate and can be several meters off. A fix for this could be using a better GPS, like an RTK GPS. Another reason they differ could be that the EKF-SLAM got the wrong initial state. The GPS and SLAM pose estimate will only match up if the initial state is very accurate. It is likely that the error seen in the figure is due to a inaccurate initial yaw.

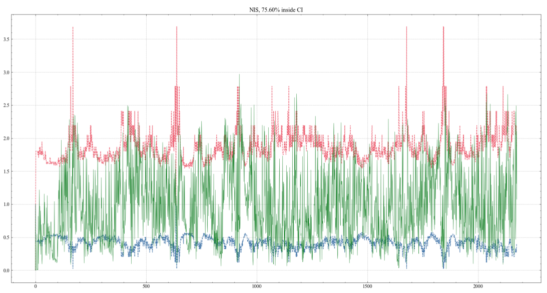

Figure 6: NIS with real data

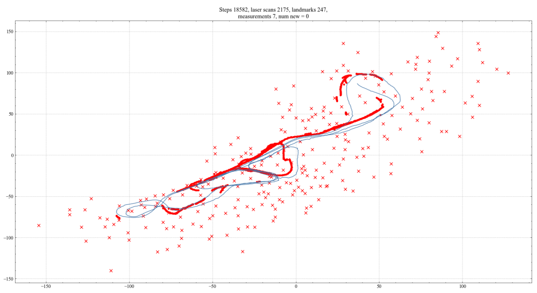

Figure 7: GPS vs estimate with real data

2.5 Alternative Algorithms

Other SLAM algorithms have managed to find solutions to many of EKF-SLAMs shortcomings. FastSLAM solves the full slam problem using particle filters and avoided storing and inverting the dense posterior covariance matrix P_k. FastSLAM is good at handling non linearities, but can be computationally heavy with many particles. Its time complexity is $\mathcal{O}(N \log M)$, where N is the number of particles and M is the number of landmarks. This time complexity is better than EKF-SLAMs $\mathcal{O}(M^2)$ if the number of particles are kept reasonably small, which could be done with Rao-Blackwellization.

ORB SLAM is a graph based full SLAM solution which creates a sparser essential graph from the dense covisibility graph. By only keeping a subset of edges (correlations) with high covisibility, a sparser graph is created. The pose graph optimisation is ran on this sparse essential graph improving the time complexity significantly. Inertia measurement's were not included in this implementation of the EKF-SLAM, but may have been able to improve the results. VINS mono (2017), VINS fusion (2019) and ORB SLAM 3 (2020) are some recent examples of VISLAM, which uses inertial measurements to improve results.