Aalto University
School of Electrical
Engineering

ELEC-E8105

NON-LINEAR FILTERING AND PARAMETER ESTIMATION

Project Report: Localization of Autonomous Cars

Author:
Aravind Swaminathan

Student Number:
769383

April 12, 2020

Contents

1 Introduction:

Autonomous vehicles have several advantages over the infrastructure changes in society, mainly increased safety. Traffic safety is very much important now a days and the data shows that no of accidents with the future of autonomous mobility will drastically reduce. This can be done by letting technology assist drivers by automating the driving task.

A driver less car will have multiple problems to be handled at the same time. This Project is based only on the localization of the driver less car. The localization of the car includes state estimation of the ego vehicle(hereafter referred to driver less car). The general flow diagram of ego vehicle will be as follows:

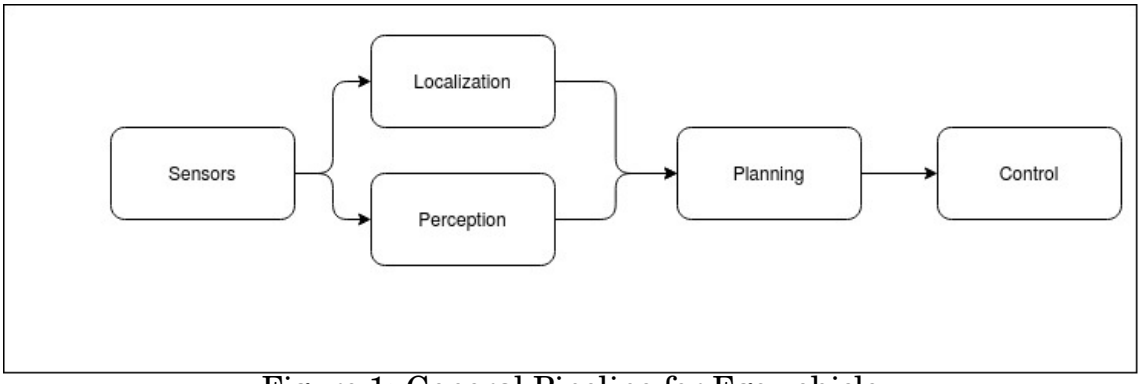

Figure 1: General Pipeline for Ego vehicle

As we can see accurate localization with respect to a map is a key enabler for using information from those maps in the planning and control module[2]. Also the positioning of the system can be obtained from various algorithms. This project will show the use of Extended Kalman filter in fusing the information from GPS, IMU , Wheel Encoders to provide an accurate position output.

1.1 Introduction of Extended Kalman Filter:

Generally Kalman filter is a state estimation technique which follows Bayesian methods to compute the state of system . Kalman filter works with Linear system models. This has some restriction with the fact that all the systems cannot be only with linear system dynamics all the time. So, Extended Kalman Filter removes the restriction of linear state transition and measurement models[3]. Instead it allows you to use any kind of nonlinear function to model the state transition and the measurements you are making with your robot. In order to still be able to use the efficient and simple Linear Algebra magic in our filter, we do a trick: We linearize the models around the current

robot state. This means that we assume the measurement model and the state transition model to be approximately linear around the state at which we are right now. While this approach forces us to make a linearization of this nonlinear function after every time step, it turns out to be not computationally expensive[1].

2 Project Description:

This project includes simulation of an ego vehicle in a simple map which was available online in gazebo simulation maps. The real-time simulation of Car model, Sensor models and the corresponding system dynamics will help us get the accurate positioning of the ego vehicle in the simulated environment.

2.1 Platforms Used:

2.1.1 ROS:

" ROS, an open-source robot operating system. ROS is not an operating system in the traditional sense of process management and scheduling; rather, it provides a structured communications layer above the host operating systems of a heterogeneous compute cluster" [8], Definition of ROS as declared in the paper. Also more information about ROS can be found in the link here ROS_LINK.

2.1.2 Gazebo:

Gazebo is a well-designed simulator as it makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. More information about Gazebo simulation can be found in the link here GAZEBO_LINK.

2.1.3 Rviz:

Rviz, abbreviation for ROS visualization, is a powerful 3D visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information. By visualizing what the robot is seeing, thinking, and doing, the user can debug a robot application from sensor inputs to planned (or unplanned) actions. More information about Rviz can be found in the link here RVIZ_LINK.

2.1.4 Programming Language:

The best programming language that can be used for production level codes is C++. So in this project the implementation of Car model, Sensor Models, Simulation environment and localization module which includes the EKF algorithms are implemented using C++.

2.2 Sensors and Car Physical Model:

2.2.1 Car Model:

Car Model: Hyundai Veloster
Car Length : 2.63m
Car Width : 1.56m
Car Steering Ratio: 15.06°

2.2.2 GPS:

GPS model: Gazebo GPS
GPS Attached Position: Rear Axle of Car
GPS Rate: 1Hz(1 secs)

2.2.3 IMU:

IMU model: Gazebo IMU
IMU Attached Position: Rear Axle of Car
IMU Rate: 100Hz(10 milliseconds)

2.2.4 Wheel Encoders:

Wheel Encoders model: Gazebo Wheel Encoders
Wheel Encoders Attached Position: All 4 wheels of Car
Wheel Encoders Rate: 100Hz(10 milliseconds)

2.3 System State Space Representation:

2.3.1 State Space model:

The system is assumed as non linear system. The general state space model of the non-linear system is given below. The system is assumed to be modelled with Gaussian noises.

$$x_k = f(x_{k-1}) + q_{k-1}$$

$$y_k = H_k * x_k + r_k$$

where,
x_k - represents the state of system
y_k - represents the measurement from sensors
q_{k-1}- Gaussian Process noise $\sim N(0, Q)$
r_{k-1}- Gaussian measurement noise $\sim N(0, R)$
$f(x_k)$- Non linear system dynamic model
H_k - Measurement Matrix

2.3.2 State Matrix:

Accurate position of the ego vehicle can be represented in terms of states. The general parameters required for representing the robot's state are position, Velocity , acceleration and Orientation. The states are always represented with respect to some frame , most likely global frame. The state matrix chosen here for the representing the Robot's state is

$$x_k = \begin{bmatrix} P_x & (position\ in\ x) \\ P_y & (position\ in\ y) \\ P_z & (position\ in\ z) \\ V_x & (velocity\ in\ x) \\ V_y & (velocity\ in\ y) \\ V_z & (velocity\ in\ z) \\ A_x & (acceleration\ in\ x) \\ A_y & (acceleration\ in\ y) \\ A_z & (acceleration\ in\ z) \\ \gamma & (Roll) \\ \beta & (Pitch) \\ \alpha & (Yaw) \\ \gamma_{rate} & (Roll\ rate) \\ \beta_{rate} & (Pitch\ rate) \\ \alpha_{rate} & (Roll\ rate) \end{bmatrix}$$

2.3.3 System matrix:

The system matrix elements shown below are only concentrated in computing the Position, velocity , acceleration and orientation. The system dynamics matrix is as follows. (Matrix information also available in appendix A)

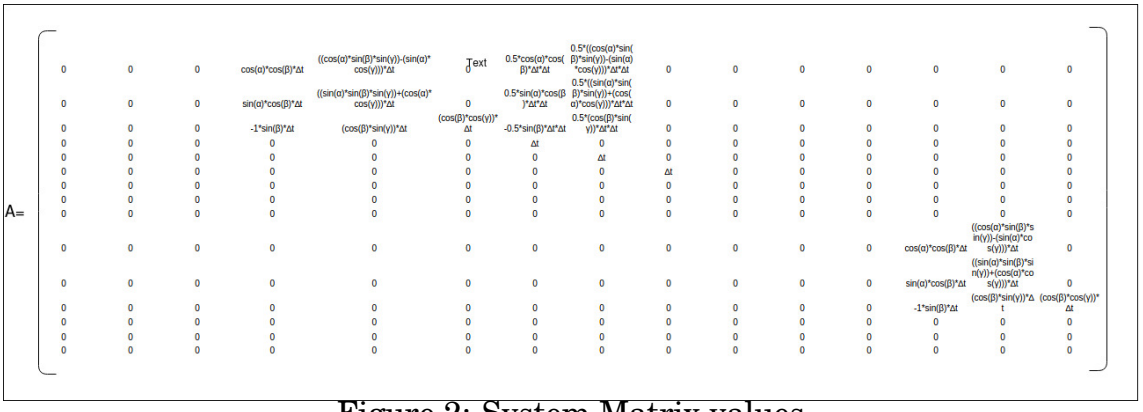

Figure 2: System Matrix values

2.3.4 Noise model Matrices:

The Noise matrices are modelled from simulator and also the assumed values are with the Gazebo's model.(Matrix information also available in appendix A)

The Process noise is

$$Q = \begin{bmatrix}
.0001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & .0001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & .0001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .001 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .0001 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0001 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0001
\end{bmatrix}$$

Figure 3: Q Matrix values

The Measurement noise is given by

$$R = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .039047 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .2795 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.080243 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0001 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0001 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0002 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0002 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0002 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0001
\end{bmatrix}$$

Figure 4: R Matrix values

2.4 Extended Kalman Filter:

This is the estimation technique to estimate the state matrix mentioned above. The estimation method involves a two step process to determine the states which are Prediction and the Update steps. The system state is first Predicted using the system dynamics in the Prediction step and then with the help of measurement the estimated state will be corrected accordingly. Also during the process of Update step, the Kalman gain which is the major factor for final estimated output will be updated every frame in this step. As mentioned earlier, the First order Taylor series expansion is used to linearize the non linearity in the system[4].

2.4.1 Linearization:

Let us assume that the an equation is of form

$$y = g(x) + q$$

where x - N(m,P)
$g(.)$ - non linear function

then the the system can be linearized at the mean using taylor series expansion as follows:

$$g(x) \approx g(m) + G_x(m)(x - m)$$

where g(m) - system's response at mean m
$G_x(m)$ - Jacobian Matrix

2.4.2 Prediction:

In the prediction step, we predict the state of the system using the system dynamic model.

The prediction step equations are ,

$$\hat{m_k} = f(m_{k-1})$$
$$\hat{P_k} = F_x(m_{k-1}) * P_{k-1} * F_x^T(m_{k-1}) + Q$$

2.4.3 Update:

In the update step, we use the Kalman gain to compute the final estimate of state taking into account the measurement information from sensors as well.

The Update step equations are,

$$K_k = \hat{P_k} * H_k^T * (H_k * \hat{P_k} * H_k^T + R)$$

$$m_k = \hat{m_k} + K_k * (y_k - H_k * \hat{m_k})$$
$$P_k = \hat{P_k} - K_k * (H_k * \hat{P_k} * H_k^T + R) * K_k^T$$

2.4.4 Localization module structure:

The localization module includes the Sensor capture block where the sensor data from GPS, IMU, wheel encoders are obtained and then the data is transformed to local frames for easy calculation purposes. The GPS data we get will be in Global coordinate system(latitude and longitude) and IMU data from the sensor can be assumed as local coordinate system. So to follow a unique coordinate system, transformations of data is first carried out, in order to fuse the information and produce a final estimate output. Here in this project, the common frame chosen is Odom frame. Odom frame represents the coordinate system which assumes the initial position as origin and there after the following convention is shown iin Figure 5.

The data which will be measured from GPS are P_x,P_y,P_z and the data which will be measured from IMU are $Roll$, $Pitch$ and Yaw

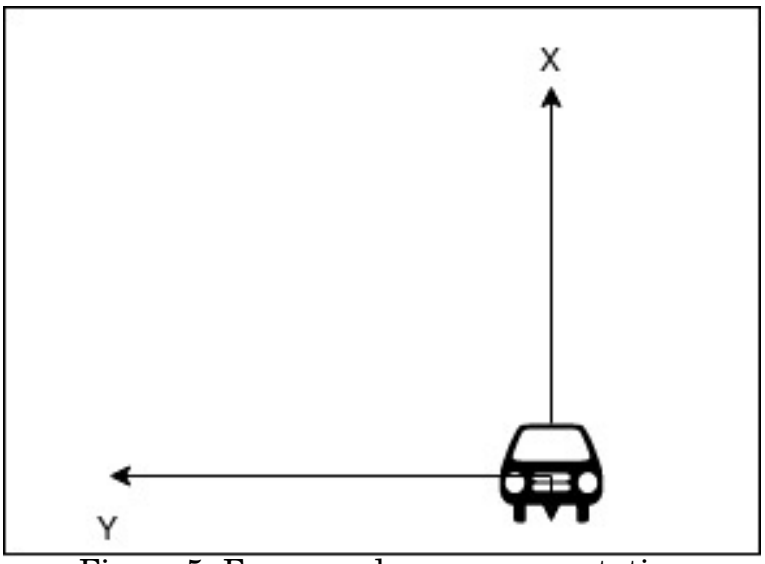

Figure 5: Frame and axes representation

The wheel encoders provide the information of wheel speed information individually. These individual wheel encoder data is then combined to get a wheel speed value which depicts V_x,$V_y$and V_z

The complete block of the localization module is as follows:

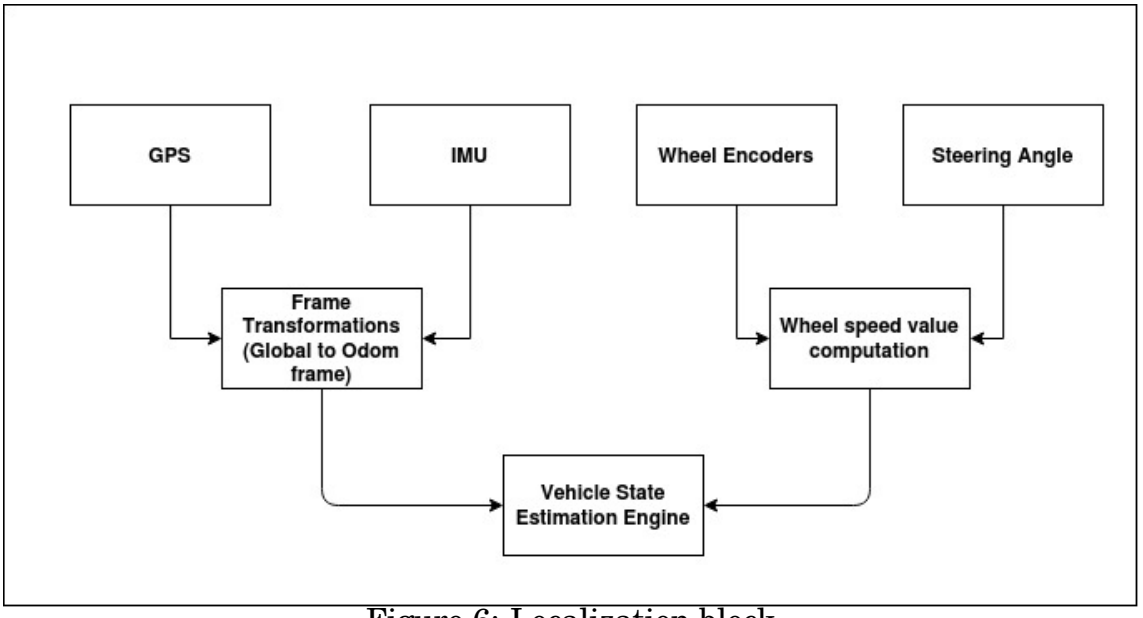

Figure 6: Localization block

3 Simulation/Results:

The simulations were carried out with Gazebo from different track positions. For each track position the result is shown correspondingly with its measurements, estimate and true path of the car. The measurements are computed from the GPS signals which has been modelled with noise, while the estimates are computed from the extended kalman filter and ground truth data will be from simulator. The location of ego vehicle is marked with green circle and the red cross indicates the destination up to which the EKF algorithm was tested.

3.1 Various Track Results :

The simulation of this Extended kalman filter in various track roads will show us a better understanding how the fusing of data between sensors for localization will help us give the accurate position of the ego vehicle. The EKF was simulated in multiple track environment and below are shared some of the few results which will show the EKF's performance in improving the positioning.

3.1.1 Test Track 1 simulation results:

The track 1 simulation can be seen below in the Gazebo simulator's window and also the corresponding Kalman Filter's response for the sensor system used. The computed RMSE value for this track is 0.3346

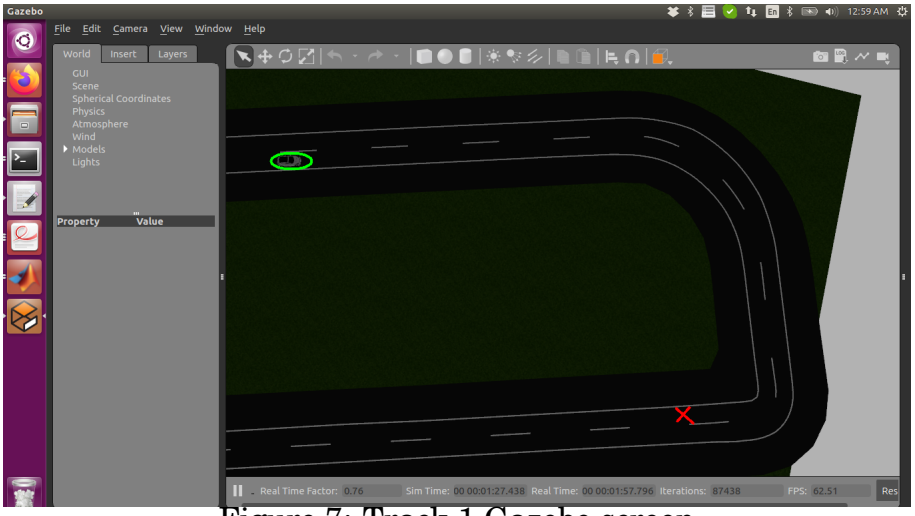

Figure 7: Track 1 Gazebo screen

Apart from position of the vehicle, the ego vehicle's state velocity can be shown as below:

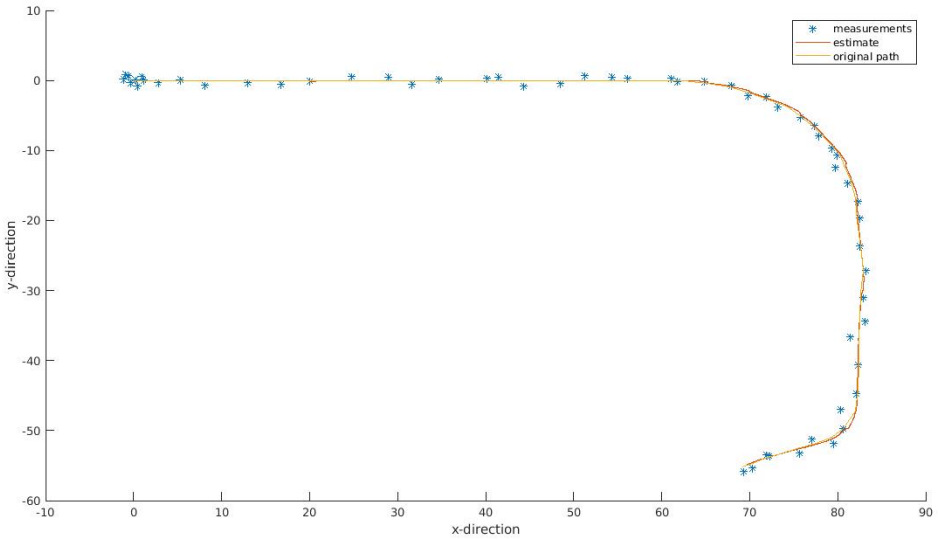

Figure 8: EKF performance in track 1

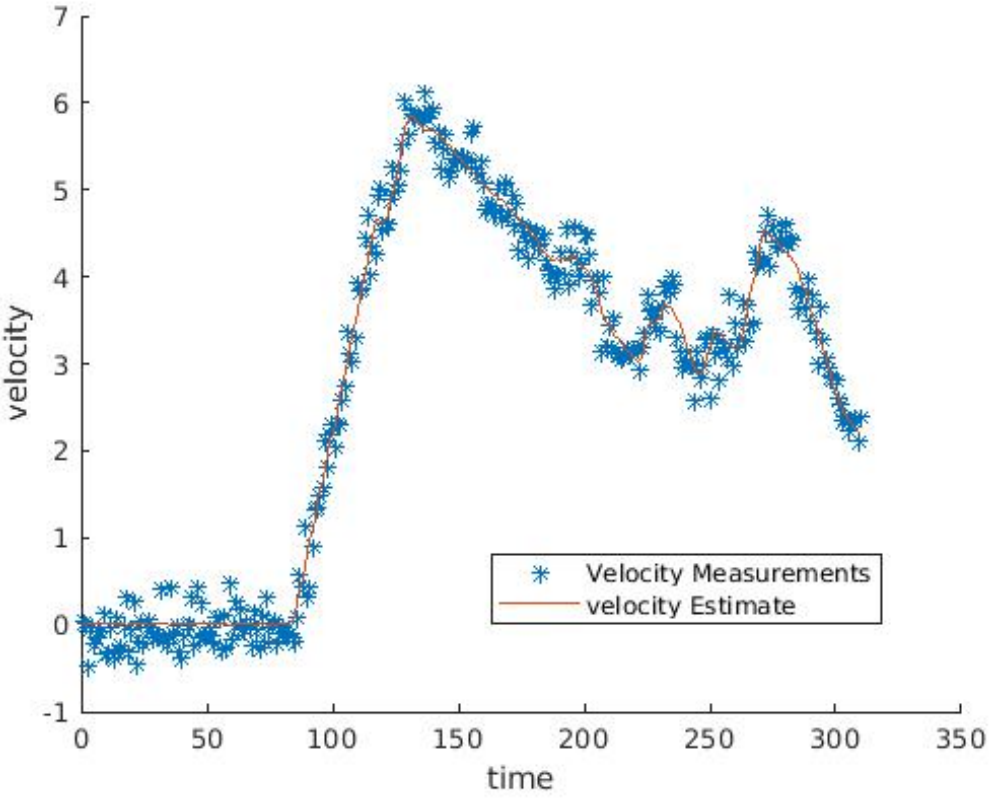

Figure 9: EKF estimate for Velocity in track 1

3.1.2 Test Track 2 simulation results:

The track 1 simulation can be seen below in the Gazebo simulator's window and also the corresponding Kalman Filter's response for the sensor system used. The computed RMSE value for this track is 0.2343

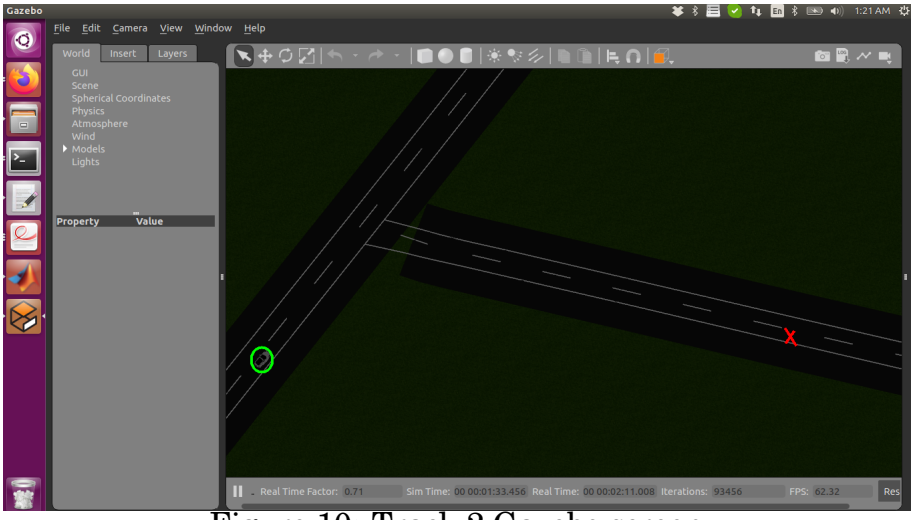

Figure 10: Track 2 Gazebo screen

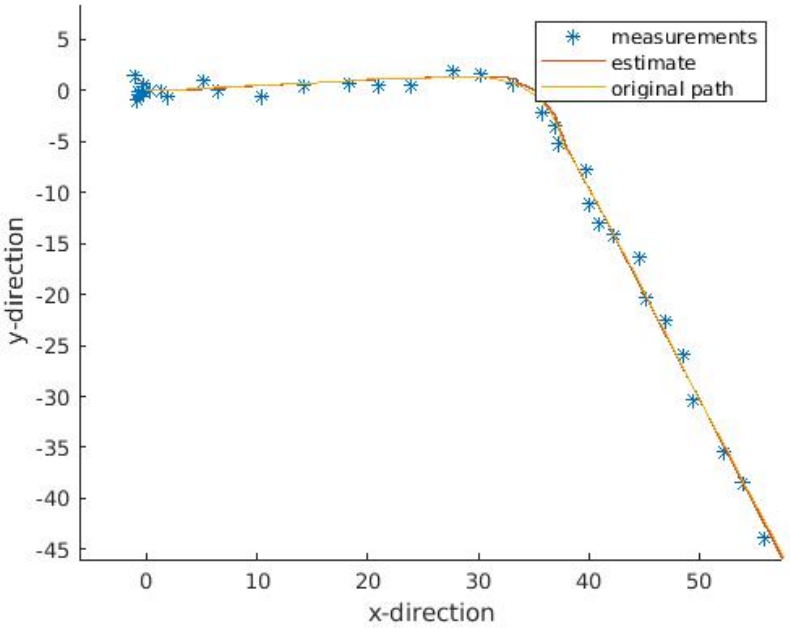

Figure 11: EKF performance in track 2

The Velocity estimate can be shown as follows:

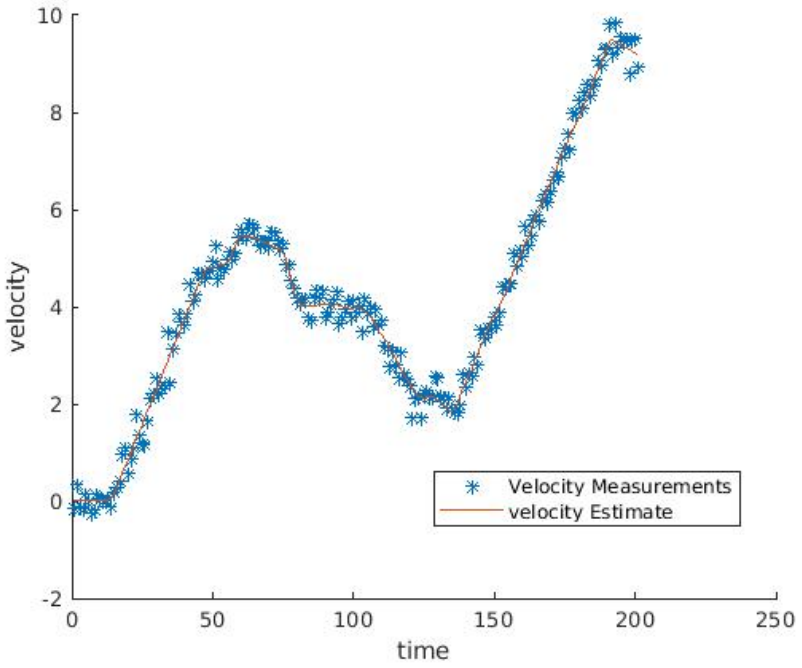

Figure 12: EKF estimate for velocity in track 2

4 Conclusion:

The future of Mobility is advancing at a faster rate and autonomous cars will have a huge market in future. As we can see the sensors that we use in real time are not completely accurate to estimate the position , velocity and orientation of the system. This causes a huge problem in decision making process of intelligent robots. To solve this issue, we need proper estimation methods and sensor fusion techniques to reduce the error in the initial modules of whole system. One such method is using Extended Kalman filters for vehicle state estimation(VSE). The process of state estimation includes acquiring sensor data from different sensors, fuse the information, remove high frequency errors and estimate the state of system.

4.1 Reasons for fusion engine:

1. The sensor data in real time scenarios will be erroneous due to real time environmental effects[**?**].For example, data from IMU will be more error prone due to its drift effect, Also the magnetometer will have more effects on any system. To reduce the high frequency error in the system, kalman filtering techniques are used to estimate the position of the robot accurately with a maximum accuracy of 15cm.
2. Data from individual sensors will not be enough to estimate the whole state of the system. So sensor fusion will help us to fuse the information from different sensors to compute the final state of the system. Kalman filtering techniques plays a major role in sensor fusion.

3. The data from individual sensors will be at a slower rate in real time scenarios, say 1Hz . But the whole system requires the localization information at faster rate to take the real time decisions. This can be achieved by using the kalman filtering based estimation techniques to predict the state[6].

4.2 Advantages of EKF in localization:

1. It is computationally efficient , as the linearization is not a costly process. It helps us produce the estimate of system in 100Hz(10ms)
2. The approximations are really good in real time scenarios
3. The computation of the theoretical equation is very simple.
4. Easy to Program and easy to debug the system.

4.3 Disadvantages of EKF in localization:

1. Doesn't work for all non-linearities where the system may not be differentiable.
2. UKF can produce better results at times where the differentiable functions are costly.
3. As the size of state matrices increase, the computation of Jacobian matrices are really costly.

4.4 Future work:

1. Implementation of Visual odometry(camera based speed detection) algorithm to efficiently estimate the velocity of the vehicle. And fuse the information using the extended kalman filter to improve better accuracy
2. Better Estimation of angular acceleration from IMU.
3. Auto calibration of sensors.
4. Better visualization in Gazebo and rviz simulation.

5 Bibliography

References

[1] Howie Choset , *Localization, Mapping, SLAM and The Kalman Filter* .[Online] Available:https://www.cs.cmu.edu/ motionplanning/lecture/Chap8-Kalman-Mapping_howie.pdf

[2] Erik Stenborg, *Localization for Autonomous Vehicles*. [Online] Available: http://publications.lib.chalmers.se/records/fulltext/252423/252423.pdf

[3] Tairan Chen et all, *FROM PERCEPTION TO CONTROL: AN AUTONOMOUS DRIVING*. [Online] Available:https://arxiv.org/pdf/1909.00119.pdf

[4] Simo Sarkka , *Bayesian Filtering and Smoothing*, [Book] Available: www.cambridge.org/sarkka

[5] Yan Wang et all, *A Fusion Localization Method based on a Robust xtended Kalman Filter and Track-Quality for Wireless Sensor Networks*, July 2019

[6] Chen et all, *Mobile Location Estimator in a Rough Wireless Environment Using Extended Kalman-Based IMM and Data Fusion.* IEEE Trans. Veh. Technol. 2009, 58, 1157–1169.

[7] Pengpeng Chen et all, *Modified Extended Kalman Filtering for Tracking withInsufficient and Intermittent Observations*, [Online] Available:http://downloads.hindawi.com/journals/mpe/2015/981727.pdf

[8] Quigley, Morgan, et al. “ROS: an open-source Robot Operating System.” ICRA workshop on open source software. Vol. 3. No3.2. 2009.

6 Appendix A

The matrices in the state space equations are given in the link below

System Matrix: System Matrix Link

R Matrix : R_Mat_link

Q Matrix: Q_mat_link

7 Appendix B

The code can be found in following link: GITHUB LINK

Code execution steps:

Pre requisites:

1. ROS installation: http://wiki.ros.org/kinetic/Installation 2. Install the GPS deamon on the system

```
sudo apt-get install libgps-dev
sudo apt-get install gpsd gpsd-clients python-gps
```

3. Clone the Github code base given in Appendix B.

```
git clone https://github.com
/aravindSwamy94/Localization-for-Autonomous-cars.git
```

4. Building the code

```
cd Catkin_Ws/
catkin_make
source devel/setup.bash
```

5. To run the simulation and to run the Localization module:

```
roslaunch simulation simulation.launch
```

To launch the localization module: open another terminal and follow:

```
source devel/setup.bash
roslaunch extended_kalman_filter vse_ekf.launch
```

To control the vehicle in simulator open another terminal and follow:

```
source devel/setup.bash
roslaunch simulation keyboard.launch
```

6. To visualize the real time localization data open another terminal and follow:

```
rviz
```

load the config file using following steps give open config from tool bar config file can be found in

`Catkin_Ws/src/extended_kalman_filter/cfg/rviz/ekf_visualization.rviz`