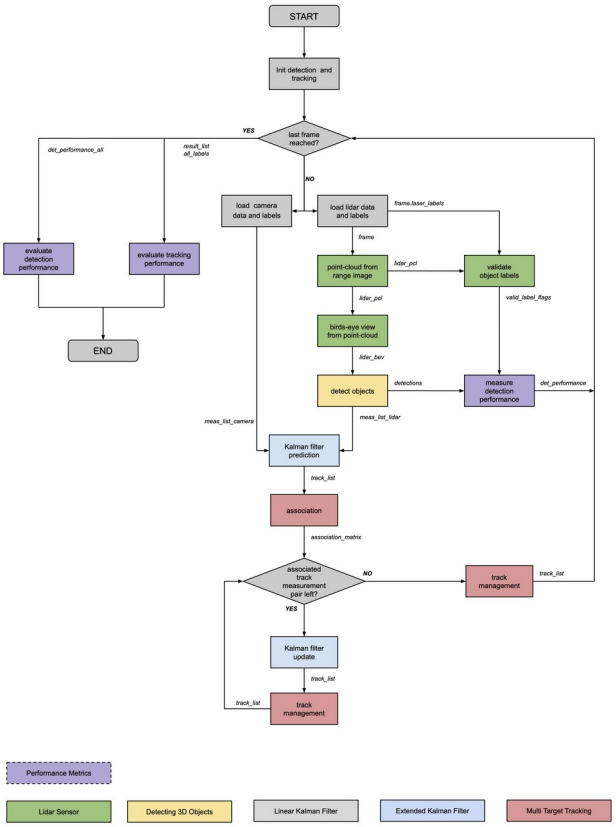
SDCND : Sensor Fusion and Tracking

Final Project Write up -

Goal of the final project is to implement Lidar and Camera sensor fusion and object tracking , track management using Waymo Open Dataset's real-world data.

The final project implements four main steps as required by the rubric. Below diagram shows the sensor fusion and object tracking algorithm.

- Step 1: Implement an extended Kalman filter.
- Step 2: Implement track management including track state and track score, track initialization and deletion.
- Step 3: Implement single nearest neighbor data association and gating.
- Step 4: Apply sensor fusion by implementing the nonlinear camera measurement model and a sensor visibility check.



Step 1: Implement an extended Kalman filter.

The goal of this task is to complete the implementation if EKF in the filter.py file for constant velocity process model in 3D with timestep dt.

Below functions are implemented in filter.py.

`Implement predict() function`
`calculate a system matrix F()`
`process noise covariance Q()`

`Implement` the `update()` function
r`esidual gamma gamma()`
`residual covariance S()`

Kamlan Filter Definitions

- x = estimate
- P = uncertainty covariance
- F = state transition matrix
- u = motion vector
- z = measurement
- H = measurement function
- R = measurement noise
- I = identity matrix

Kalman Filter Prediction Equations

$x' = Fx + u$

$P' = F \cdot P \cdot FT$

Measurement Equations

$y = z - H \cdot x$

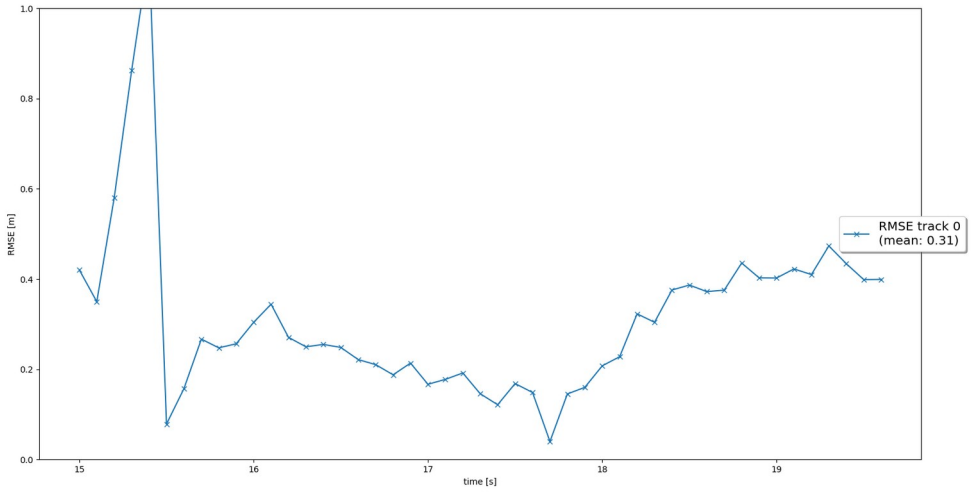
$S = H \cdot P \cdot HT + R$

$K = P \cdot HT \cdot S - 1$

$x' = x + (K \cdot y)$

$P' = (I - K \cdot H) \cdot P$

Below RMSE graph is generated for a single vehicle , for the given the project setup instructions . RMSE < 0.35 in this setting ensures correct implementation of the EKF.



Step 2: Implement track management including track state and track score, track initialization and deletion.

Track initialization -
In trackmeasurement.py the track constructor takes in a sensor measurement for unassigned track to create a new track. The sensor measurement is transformed from sensor to vehicle coordinates and stored on the position vector x.
Similarly the uncertainty co-variance vector P is initialized with following equations given high uncertainty in the velocity estimation.

$$\mathbf{P}_{\text{pos}} = \mathbf{M}_{\text{rot}} \cdot \mathbf{R} \cdot \mathbf{M}_{\text{rot}}^{T}$$

$$\mathbf{P}_{0} = \left(\begin{array}{c|c} \mathbf{P}_{\text{pos}} & 0 \\ \hline 0 & \mathbf{P}_{\text{vel}} \end{array} \right)$$

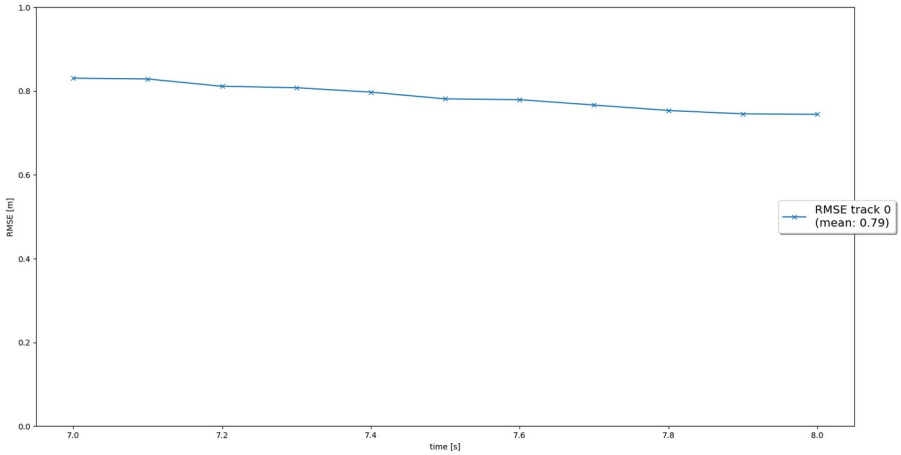
```
# decrease score for unassigned tracks

track.score -= 1./params.window

# delete old tracks
```

The old tracks are deleted with two different threshold levels. For previously confirmed the threshold used is `threshold = params.delete_threshold= 0.6` and the new or tentative tracks threshold = `0.5/params.window.`

Also the tracks with very high position uncertainty are deleted irrespective of their state and score.



Step 3: Implement single nearest neighbour data association and gating.

Mahalanobis distances between all the targets and measurements are calculated using below Mahalanobis distance formula.

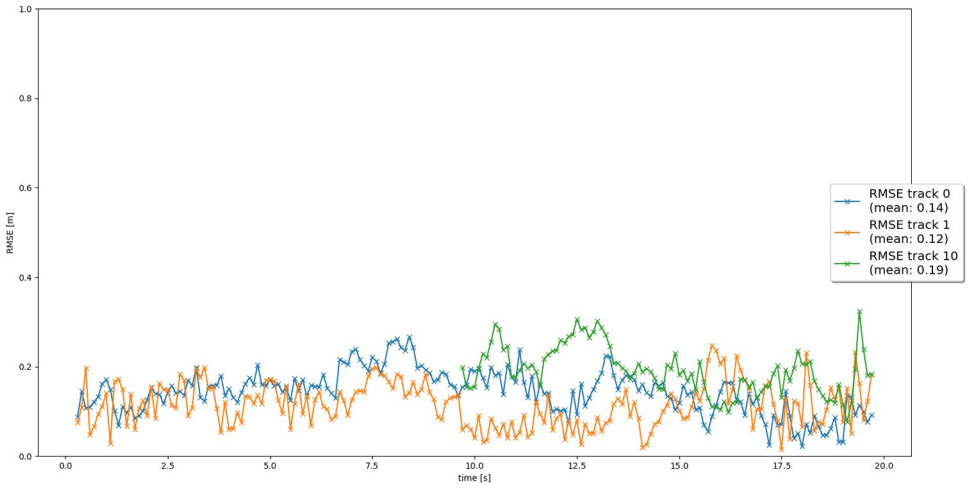
$$d(\mathbf{x}, \mathbf{z}) = \gamma^T \mathbf{S}^{-1} \gamma = (\mathbf{z} - h(\mathbf{x}))^T \mathbf{S}^{-1} (\mathbf{z} - h(\mathbf{x}))$$

Gating Function – returns if the the MHD falls within the track's gate. The association matrix is only updated if the MHD returns true.

$$d\left(\mathbf{x}, \mathbf{z}\right) \leq F_{\chi^2}^{-1}\left(0.995 | \dim_{\mathbf{z}}\right)$$

Association matrix A – is initialized with inf values for all meas and track pairs and updated with corresponding MHD. We are using Simple Nearest Neighbor (SNN) data association to match the measurement – target pair. Unpaired measurements and unpaired tracks are stored in

```
self.unassigned_tracks = []
self.unassigned_meas = []
```



Step 4: Apply sensor fusion by implementing the nonlinear camera measurement model and a sensor visibility check.

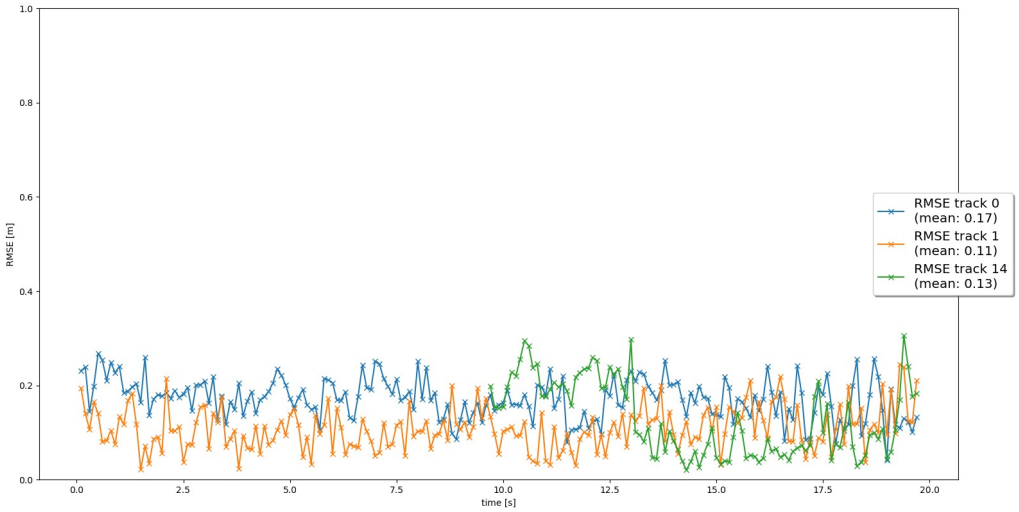
`student/measurements.py implements the required camera measurement model for this step.`

`in_fov() that checks if the input state vector x of an object can be seen by this sensor and the score is decreased only if in_fov() is true.`

`Measurement function get_hx() is updated with hx for camera non linear measurement.`

`Finally the camera measurement is enabled in the file for final simulation.`

The results of the fusion tracking is shown in the below image and results/my_tracking_results.avi file.



Do you see any benefits in camera-lidar fusion compared to lidar-only tracking (in theory and in your concrete results)?

Although lidar can high position accuracy readings , it can present limitations when it comes to highly absorptive black images. These objects may go undetected due to low or no lidar returns. As well for highly reflective small objects may lead to false positives.
A camera gives no depth information, has poor performance in low light conditions .But it is best suited for object detection. The text on traffic sign can only be detected with camera based image detection.
To overcome limitations , reliability issues presented by single sensor like blockage or misscalibration we need to consider input from multiple sensors. This will lead to higher tracking accuracy and reliability for the self driving car.

Looking at both the tracking with lidar only and camera + lidar fusion , we can clearly see that more number of objects are detected with fusion systems. The final implementation keeps only long stable tracks.

Which challenges will a sensor fusion system face in real-life scenarios? Did you see any of these challenges in the project?

A lot of work in this project is targeted towards classroom learning and we are using adapted/prepossessed data and results. Actual real-life scenarios and real-life data will need a lot of process power , models may needed to retrained , camera and lidar calibration accuracy will play a role in final outcome. We may need to consider input from other vehicle sensors like odometer , IMU etc.

The challenging part for me was during step 4 , after adding the camera measurements , the tracking started behaving weirdly. For frame no 182 I was getting a MHD of 0.57 for track 1. This is where track 0 is changing the lane. The gating limit was lower around 0.39 . This was leading to association problems with track 1 and creating a new track afterwords and track 1 getting deleted eventually.
Thanks to mentor team , I was able to point the issue to gating implementation.

Can you think of ways to improve your tracking results in the future?

We have implemented a constant velocity process model in order to keep things simple and use a linear

process model. The the acceleration factor is considered as process noise. This could lead to estimation and tracking errors in scenarios where we have high acceleration or breaking rates. Real life driving is nonlinear and a better process model could be used to improve tracking further.

Also as stated in the classroom videos SNN is not globally optimum and more suitable Association algorithm can be used such as Global Nearest Neighbor (GNN) or Probabilistic Data Association (PDA)