Sensor Fusion and Object Tracking

1 Introduction

This project consists of four steps

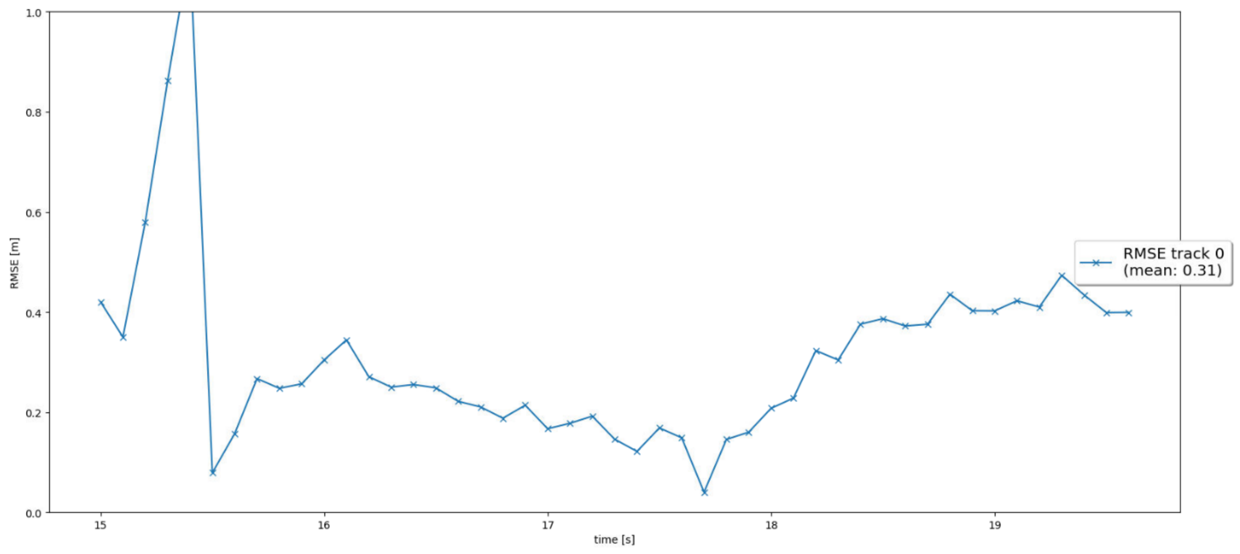
- Step 1: Implement an extended Kalman filter.
- Step 2: Implement track management including track state and track score, track initialization and deletion.
- Step 3: Implement single nearest neighbor data association and gating.
- Step 4: Apply sensor fusion by implementing the nonlinear camera measurement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

2 Step 1: Extended Kalman Filter (EKF)

The goal with this step is to create and EKF to track a single real-world target with lidar measurement input over time.

Implemented predict() and update() function for the EKF. Completed F() and Q() functions to calculate a system matrix for constant velocity process model in 3D and the corresponding process noise covariance depending on time step dt. Coded gamma() and S() functions for residual and residual covariance.

The output from the implemented EKF can be seen below. It does a good job at keep the RSME low. A lower RSME always desired. This could be achieved by tuning the process noise design parameter.

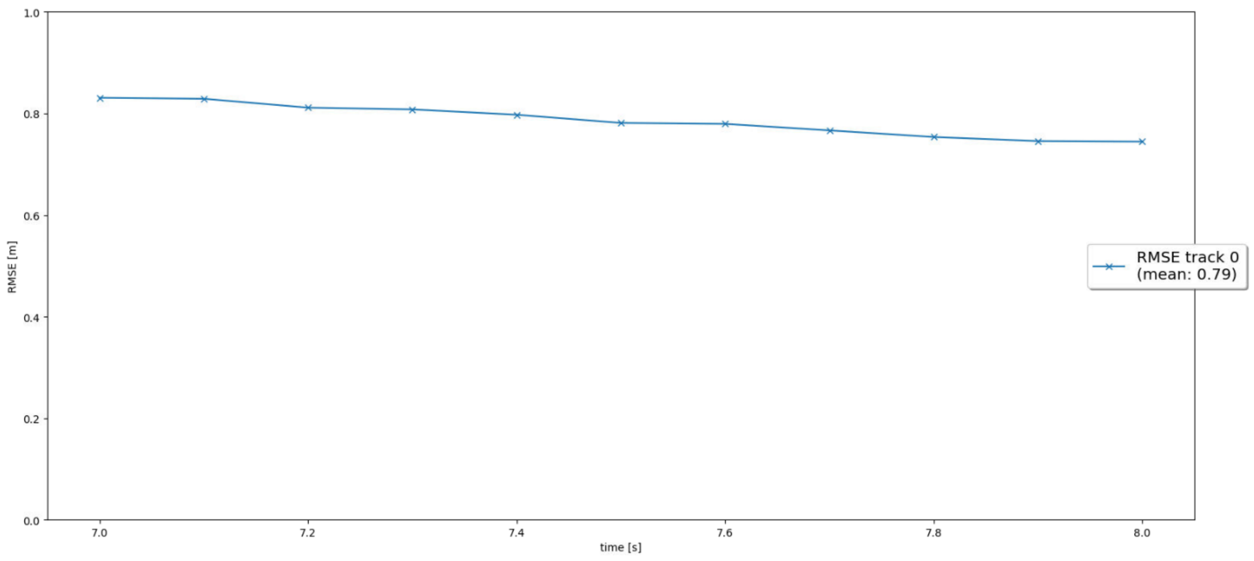


3 Step 2: Track Management

The goal here is to implement the track management to initialize and delete tracks, set a track state and a track score.

In Trackmanagement class, function manage_tracks() was implemented to decrease track score of unassigned tracks and delete tracks if the score was too low or covariance was too high. Function handle_updated_track() was also implemented to increase track score for a track by a fixed amount and set the track state as 'tentative' or 'confirmed' given the track score.

The output after the implementation of track management is below. The RSME does increase when we added track management for a single tracked vehicle. However, we expect this to reduce as we perform the next steps of association and sensor fusion.

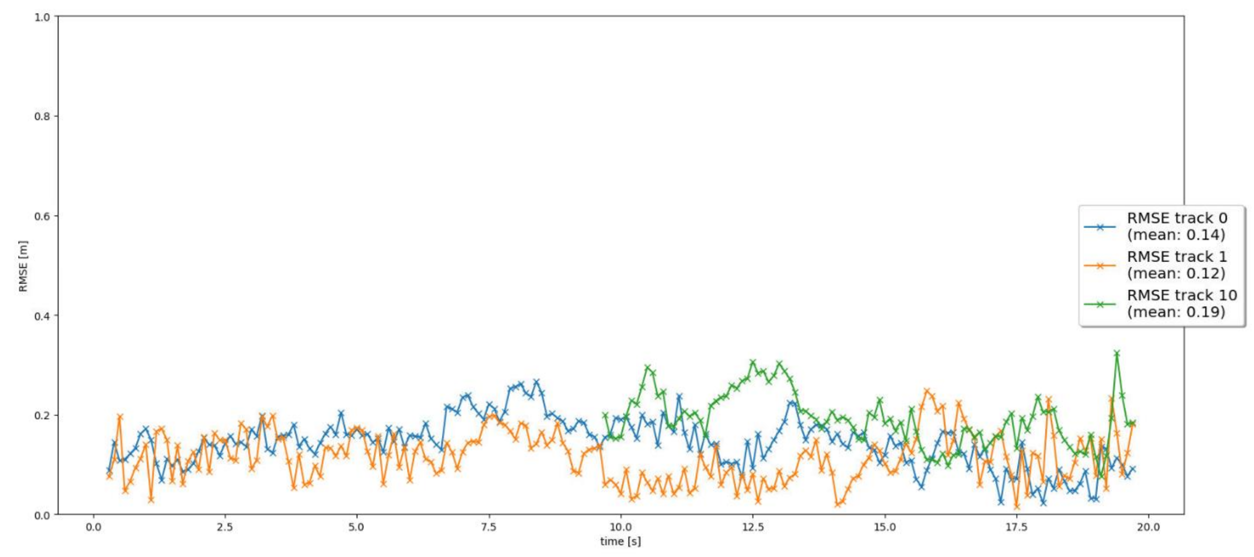


4 Step 3: Association

The goal in this step is to implement a single nearest neighbor data association to associate measurements to tracks. This moves the code into multi tracking domain.

An association matrix based on Mahalanobis distances for all tracks and measurements was created. A gating function was also used to check if a measurement lies within a track's gate. Finally an updated list of unassigned tracks and measurements were extracted. Function get_closest_track_meas() was used find minimum enter in association matrix and delete the corresponding row and column of the matrix. It also removes the corresponding track and measurement from unassigned tracks and measurements. It returns association pair between track and measurement.

The output from the RSME plot can be seen below. As shown the code can track multiple targets with low error now.

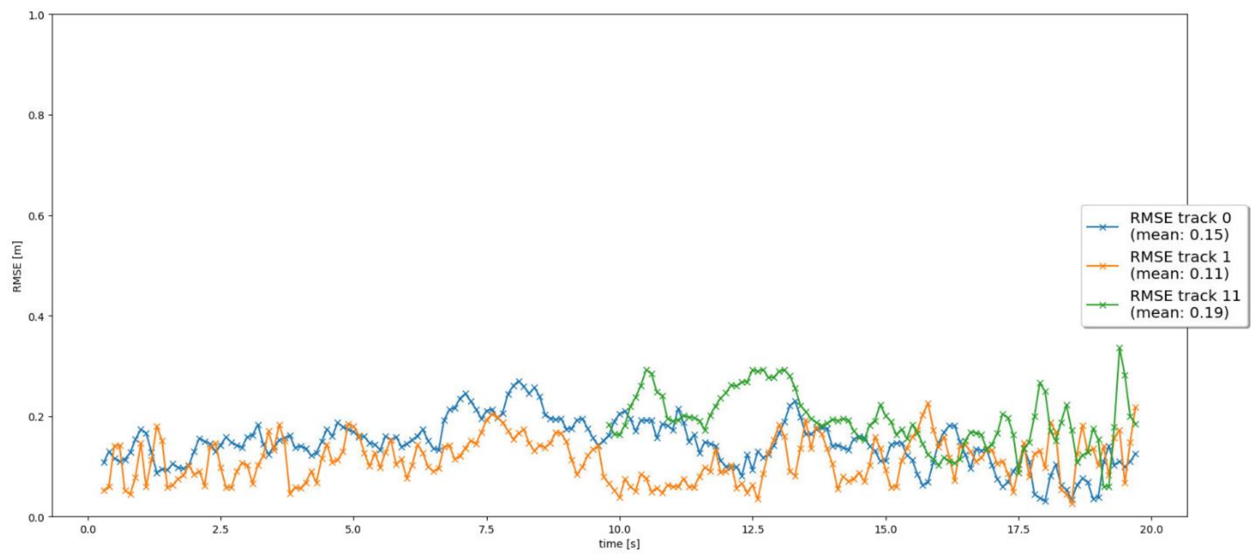


5 Step 4: Sensor Fusion

The goal in this step is to implement the nonlinear camera measurement model. Finally, the code is complete to perform the sensor fusion for camera-lidar fusion.

Function in_fov() was implemented to find if state is within sensor's fov. Implemented the function get_hx() with the nonlinear camera measurement function. In measurement class, camera measurement objects were initialized.

The output from the code can be seen below. The RSME plot shows 3 confirmed tracks all with mean RSME lower than 0.25. Thus, sensor fusion is operating as expected reducing error by using information from both lidar and camera detections.



6 Write-up Questions

6.1 Write a short recap of the four tracking steps and what you implemented there (EKF, track management, data association, camera-lidar sensor fusion). Which results did you achieve? Which part of the project was most difficult for you to complete, and why?

Most of the question is answered in section 1-5 above with the exception of what I found hard which is answered below.

The most difficult part of the project for me was debugging. I started with a code which worked until the step 3 and when I started fusing camera and lidar measurements. I kept getting an error when the track scores would keep resetting. Thus, I had to start deep-diving outside of the tasks and understand the whole structure. I eventually was able to address the error by resetting the workspace and rewriting the code. My prevailing theory is that in the changes I made for the final step I may have modified something I shouldn't have.

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

There are benefits to fusing camera and lidar measurements. It helps in re-enforcing the presence of an object around the autonomous vehicle. Fusion also helps in assigning type to lidar objects. After fusing the distance estimate from the lidar can have type associated to it where it's a car, bike, or pedestrian.

In the results after fusing lidar and camera the objects detected are more stable in tracking and have lower error in their detection. There are improvements to the detection that can only be seen in the output video where the objects being tracked were well aligned to the location of the ground truth.

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

One challenge observed with the real-world data was the difference in the FOV for camera and lidar. Lidar could detect the vehicle approaching from behind much quicker than camera. This is because lidar has a wider FOV than camera. So, the object approaching from behind was deleted a couple of times until it reaches the camera FOV. After that the tracking was stable.

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

Based on the challenge discussed in the question above, one improvement to be made would be add more cameras and lidars and fuse their information as well. This way we can have multiple sensors detecting an object approaching from behind. It will reduce the possibility of FOV causing tracking to be dropped because a sensor can't see the object yet.