

Abstract:

This project will introduce you to the different sensors and how we can use them for state estimation and localization in a self-driving car. By the end of this project, you will be able to:

- Understand the key methods for parameter and state estimation used for autonomous driving, such as the method of least-squares.
- Develop a model for typical vehicle localization sensors, including GPS and IMUs.
- Apply extended and unscented Kalman Filters to a vehicle state estimation problem.
- Understand LIDAR scan matching and the Iterative Closest Point algorithm.
- Apply these tools to fuse multiple sensor streams into a single state estimate for a self-driving car.

In this assignment, you will implement the Error-State Extended Kalman Filter (ES-EKF) to localize a vehicle using data from the CARLA simulator. The project has three parts, which should be completed in sequence:

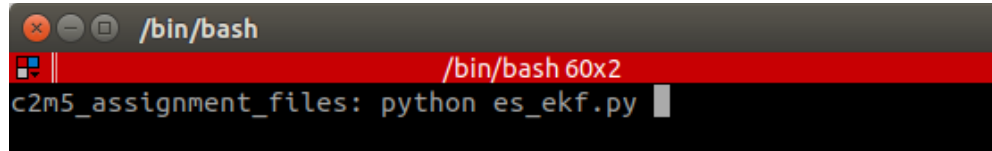
1. First, you will fill in the skeleton implementation of the ES-EKF that is provided, by writing code to perform the filter prediction step and the correction step. The filter relies on IMU data to propagate the state forward in time, and GPS and LIDAR position updates to correct the state estimate. For **Part 1** of the project, the sensor data have been prepackaged for you - it is possible to visualize the output of the estimator and compare it to the ground truth vehicle position (the ground truth position data are also provided). Your complete filter implementation will be tested by comparing the estimated vehicle position (produced by your code) with the ground truth position, for a 'hold out' portion of the trajectory (i.e., for which ground truth is not provided to you).
2. In **Part 2**, you will examine the effects of sensor miscalibration on the vehicle pose estimates. Specifically, you will uncomment a block of code that intentionally alters the transformation between the LIDAR sensor frame and the IMU sensor frame; use of the incorrect transform will result in errors in the vehicle position estimates. After looking at the errors, your task is to determine how to adjust the filter parameters (noise variances) to attempt to compensate for these errors. The filter code itself should remain unchanged. Your updated filter (with the new parameter(s)) will be tested in the same way as in Part 1.
3. In **Part 3**, you will explore the effects of sensor dropout, that is, when all external positioning information (from GPS and LIDAR) is lost for a short period of time. For Part 3, you will load a different dataset where a portion of the GPS and LIDAR measurements are missing (see the detailed instructions below). The goal of Part 3 is to illustrate how the loss of external corrections results in drift in the vehicle position estimate, and also to aid in understanding how the uncertainty in the position estimate changes when sensor measurements are unavailable.

The repository for this project:

https://github.com/Padmanabha123/Self_Driving_Using_carla/tree/main/Part%20%20-%20State%20estimation%20project%20with%20report

There are a large number of block comments throughout `es_ekf.py`. Take a look at the comments to try to understand what each section of the code is doing, and in particular to understand all of the data contained in the Python pickle (pkl) files. As the code runs, some visualizations (plots) will already appear for you, including a plot of the ground truth trajectory, a plot of the ground truth trajectory compared to your estimated trajectory, and six error plots.

To run es_ekf.py, simply call 'python es_ekf.py' from the command line or 'run es_ekf.py' from within an interactive shell.



```
/bin/bash
/bin/bash 60x2
c2m5_assignment_files: python es_ekf.py
```

Getting Output for Submission:

In the 'Data' section of the code, you'll see the following line:

```
1 with open('student_data/pt1_data.pkl', 'rb') as file:
2     data = pickle.load(file)
```

Ensure that you leave this as pt1_data.pkl for Parts 1 and 2. For Part 3, switch it to pt3_data.pkl.

The following block of code defines the calibration rotation matrix that determines the rotation between the LIDAR sensor frame and the IMU frame.

```
1 # This is the correct calibration rotation matrix, corresponding
  to an euler rotation of 0.05, 0.05, .1.
2 C_li = np.array([
3     [ 0.99376, -0.09722,  0.05466],
4     [ 0.09971,  0.99401, -0.04475],
5     [-0.04998,  0.04992,  0.9975 ]
6 ])
7
8 # This is an incorrect calibration rotation matrix, corresponding
  to a rotation of 0.05, 0.05, 0.05
9 # C_li = np.array([
10 #     [ 0.9975, -0.04742,  0.05235],
11 #     [ 0.04992,  0.99763, -0.04742],
12 #     [-0.04998,  0.04992,  0.9975 ]
13 # ])
```

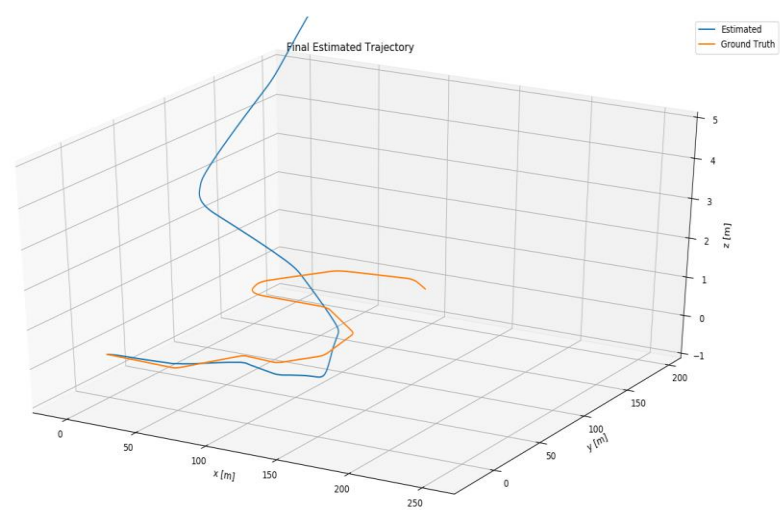
Ensure that you leave the top lines uncommented for Parts 1 and 3, but leave the bottom lines uncommented for Part 2. As well, at the bottom of the file, you'll see three blocks that look like this:

```
1 # Pt. 1 submission
2 pt1_indices = [9000, 9400, 9800, 10200, 10600]
3 pt1_str = ''
4 for val in pt1_indices:
5     for i in range(3):
6         p1_str += str(p_est[val, i]) + ' '
7 with open('pt1_submission.txt', 'w') as file:
8     file.write(p1_str)
```

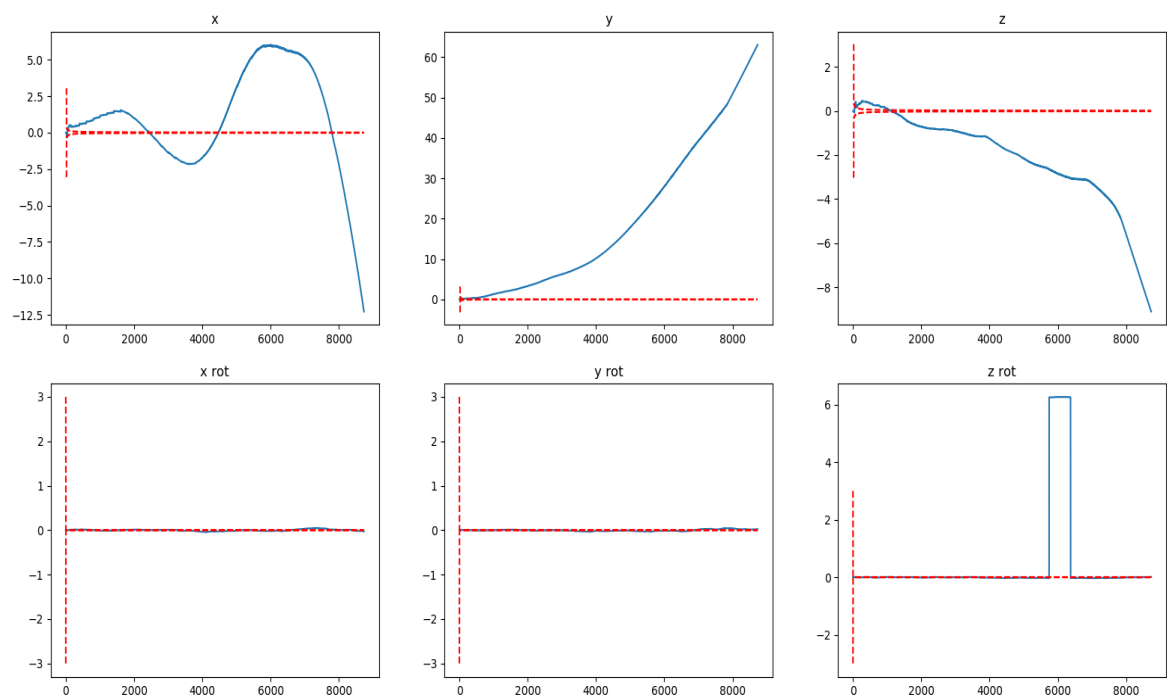
Ensure that the lines under Pt. 1 submission are uncommented for Part 1, the lines under Pt. 2 submission are uncommented for Part 2, and the lines under Pt. 3 submission are uncommented for Part 3. This will generate the right set of output data for each portion of the project

Results:

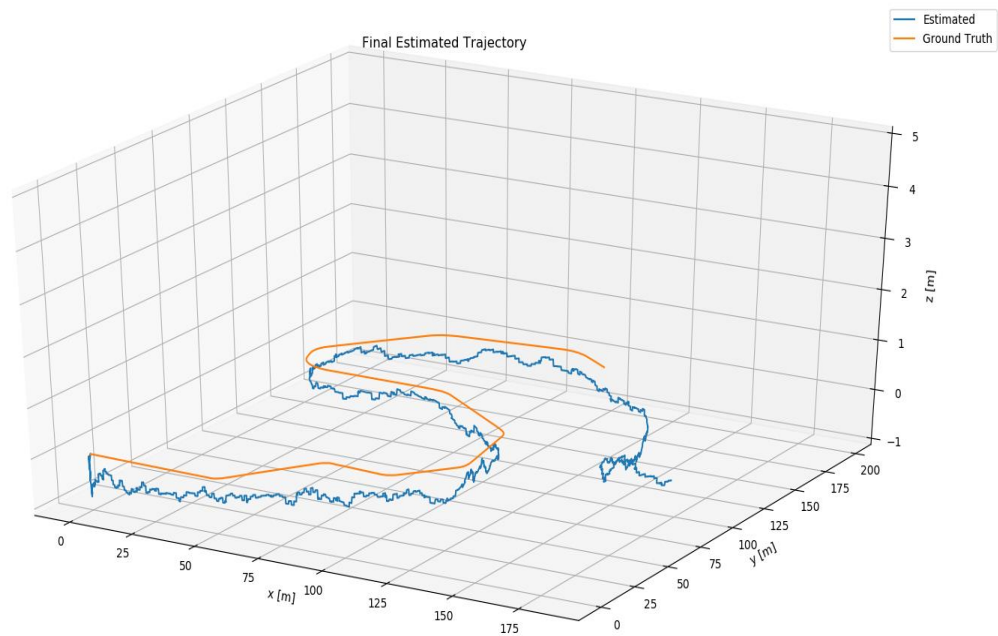
From raw IMU data:



Error plots



Acceptable error using LiDAR and GNSS data:



Error plots

