 # SDCND : Sensor Fusion and Tracking
This is the project for the second course in the [Udacity Self-Driving Car Engineer Nanodegree Program](https://www.udacity.com/course/c-plus-plus-nanodegree--nd213) : Sensor Fusion and Tracking.

In this project, you'll fuse measurements from LiDAR and camera and track vehicles over time. You will be using real-world data from the Waymo Open Dataset, detect objects in 3D point clouds and apply an extended Kalman filter for sensor fusion and tracking.

The project consists of two major parts:
1. **Object detection**: In this part, a deep-learning approach is used to detect vehicles in LiDAR data based on a birds-eye view perspective of the 3D point-cloud. Also, a series of performance measures is used to evaluate the performance of the detection approach.
2. **Object tracking** : In this part, an extended Kalman filter is used to track vehicles over time, based on the lidar detections fused with camera detections. Data association and track management are implemented as well.

The following diagram contains an outline of the data flow and of the individual steps that make up the algorithm.

Also, the project code contains various tasks, which are detailed step-by-step in the code. More information on the algorithm and on the tasks can be found in the Udacity classroom.

Project File Structure

project
┣ dataset --> contains the Waymo Open Dataset sequences
┃
┣ misc
┃ ┣ evaluation.py --> plot functions for tracking visualization and RMSE calculation
┃ ┣ helpers.py --> misc. helper functions, e.g. for loading / saving binary files
┃ ┗ objdet_tools.py --> object detection functions without student tasks
┃ ┗ params.py --> parameter file for the tracking part
┃
┣ results --> binary files with pre-computed intermediate results
┃
┣ student
┃ ┣ association.py --> data association logic for assigning measurements to tracks incl. student tasks
┃ ┣ filter.py --> extended Kalman filter implementation incl. student tasks
┃ ┣ measurements.py --> sensor and measurement classes for camera and lidar incl. student tasks
┃ ┣ objdet_detect.py --> model-based object detection incl. student tasks
┃ ┣ objdet_eval.py --> performance assessment for object detection incl. student tasks
┃ ┣ objdet_pcl.py --> point-cloud functions, e.g. for birds-eye view incl. student tasks
┃ ┗ trackmanagement.py --> track and track management classes incl. student tasks
┃
┣ tools --> external tools

```
│   ├── objdet_models --> models for object detection<br>
│   │   <br>
│   │   ├── darknet<br>
│   │   │   ├── config<br>
│   │   │   ├── models --> darknet / yolo model class and tools<br>
│   │   │   ├── pretrained --> copy pre-trained model file here<br>
│   │   │   │   └── complex_yolov4_mse_loss.pth<br>
│   │   │   ├── utils --> various helper functions<br>
│   │   <br>
│   │   └── resnet<br>
│   │   │   ├── models --> fpn_resnet model class and tools<br>
│   │   │   ├── pretrained --> copy pre-trained model file here <br>
│   │   │   │   └── fpn_resnet_18_epoch_300.pth <br>
│   │   │   ├── utils --> various helper functions<br>
│   │   <br>
│   └── waymo_reader --> functions for light-weight loading of Waymo sequences<br>
<br>
├── basic_loop.py<br>
├── loop_over_dataset.py<br>
```

Installation Instructions for Running Locally
Cloning the Project
In order to create a local copy of the project, please click on "Code" and then "Download ZIP". Alternatively, you may of-course use GitHub Desktop or Git Bash for this purpose.

Python
The project has been written using Python 3.7. Please make sure that your local installation is equal or above this version.

Package Requirements
All dependencies required for the project have been listed in the file `requirements.txt`. You may either install them one-by-one using pip or you can use the following command to install them all at once:
`pip3 install -r requirements.txt`

Waymo Open Dataset Reader
The Waymo Open Dataset Reader is a very convenient toolbox that allows you to access sequences from the Waymo Open Dataset without the need of installing all of the heavy-weight dependencies that come along with the official toolbox. The installation instructions can be found in `tools/waymo_reader/README.md`.

Waymo Open Dataset Files
This project makes use of three different sequences to illustrate the concepts of object detection and tracking. These are:
- Sequence 1 : `training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord`
- Sequence 2 : `training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord`
- Sequence 3 : `training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord`

To download these files, you will have to register with Waymo Open Dataset first: [Open Dataset – Waymo](https://waymo.com/open/terms), if you have not already, making sure to note "Udacity" as your institution.

Once you have done so, please [click here](https://console.cloud.google.com/storage/browser/waymo_open_dataset_v_1_2_0_individual_files) to access the Google Cloud Container that holds all the sequences. Once you have been cleared for access by Waymo (which might take up to 48 hours), you can download the individual sequences.

The sequences listed above can be found in the folder "training". Please download them and put the `tfrecord`-files into the `dataset` folder of this project.

Pre-Trained Models
The object detection methods used in this project use pre-trained models which have been provided by the original authors. They can be downloaded [here](https://drive.google.com/file/d/1Pqx7sShlqKSGmvshTYbNDcUEYyZwfn3A/view?usp=sharing) (darknet) and [here](https://drive.google.com/file/d/1RcEfUIF1pzDZco8PJkZ10OL-wLL2usEj/view?usp=sharing) (fpn_resnet). Once downloaded, please copy the model files into the paths `/tools/objdet_models/darknet/pretrained` and `/tools/objdet_models/fpn_resnet/pretrained` respectively.

Using Pre-Computed Results

In the main file `loop_over_dataset.py`, you can choose which steps of the algorithm should be executed. If you want to call a specific function, you simply need to add the corresponding string literal to one of the following lists:

- `exec_data` : controls the execution of steps related to sensor data.
  - `pcl_from_rangeimage` transforms the Waymo Open Data range image into a 3D point-cloud
  - `load_image` returns the image of the front camera

- `exec_detection` : controls which steps of model-based 3D object detection are performed
  - `bev_from_pcl` transforms the point-cloud into a fixed-size birds-eye view perspective
  - `detect_objects` executes the actual detection and returns a set of objects (only vehicles)
  - `validate_object_labels` decides which ground-truth labels should be considered (e.g. based on difficulty or visibility)
  - `measure_detection_performance` contains methods to evaluate detection performance for a single frame

In case you do not include a specific step into the list, pre-computed binary files will be loaded instead. This enables you to run the algorithm and look at the results even without having implemented anything yet. The pre-computed results for the mid-term project need to be loaded using [this](https://drive.google.com/drive/folders/1-s46dKSrtx8rrNwnObGbly2nO3i4D7r7?usp=sharing) link. Please use the folder `darknet` first. Unzip the file within and put its content into the folder `results`.

- `exec_tracking` : controls the execution of the object tracking algorithm

- `exec_visualization` : controls the visualization of results

  - `show_range_image` displays two LiDAR range image channels (range and intensity)
  - `show_labels_in_image` projects ground-truth boxes into the front camera image
  - `show_objects_and_labels_in_bev` projects detected objects and label boxes into the birds-eye view
  - `show_objects_in_bev_labels_in_camera` displays a stacked view with labels inside the camera image on top and the birds-eye view with detected objects on the bottom
  - `show_tracks` displays the tracking results
  - `show_detection_performance` displays the performance evaluation based on all detected
  - `make_tracking_movie` renders an output movie of the object tracking results

Even without solving any of the tasks, the project code can be executed.

The final project uses pre-computed lidar detections in order for all students to have the same input data. If you use the workspace, the data is prepared there already. Otherwise, [download the pre-computed lidar detections](https://drive.google.com/drive/folders/1IkqFGYTF6Fh_d8J3UjQOSNJ2V42UDZpO?usp=sharing) (~1 GB), unzip them and put them in the folder `results`.

External Dependencies
Parts of this project are based on the following repositories:
- [Simple Waymo Open Dataset Reader](https://github.com/gdlg/simple-waymo-open-dataset-reader)
- [Super Fast and Accurate 3D Object Detection based on 3D LiDAR Point Clouds](https://github.com/maudzung/SFA3D)
- [Complex-YOLO: Real-time 3D Object Detection on Point Clouds](https://github.com/maudzung/Complex-YOLOv4-Pytorch)

License

This is a template submission for the final project: Sensor fusion and object tracking.

Sensor Fusion and Object Tracking

We have used the Waymo dataset real-world data and used kalman filter over time to track the multi-object by sensor fusion.

- Before implementation kalman filter , it need to configure fpn_resnet model under objdet_detect.py
- EKF is implemented including appropriate system matrix F and process noise Q for constant velocity motion model under method filter.py to obtain simple signle target scenerio with only lidar and RMSE value which help to track single object.
- Under trackmanagement.py,initializing,scoring ,updating and deleting from visible range of is perform and obtain RMSE value that help to manage track of objects.
- Inside STEP3, Associate measurements to tracks with nearest neighbor associationwith RMSE plots.
- STEP4, combining sensor (lidar) and camera to measure tracks of vehicles or objects.

The project can be run by running

python loop_over_dataset.py

All training/inference is done on udacity workspace.

Step-1: Tracking

In this we are first previewing implement an EKF to track a single real-world target with lidar measurement input over time including appropriate system matrix F and process noise Q for constant velocity motion model.

- Implement Extended Kalman Filter applied to a simple single-target scenario with lidar only.
- Prediction and updation of state x
- Plot RMSE values in graph

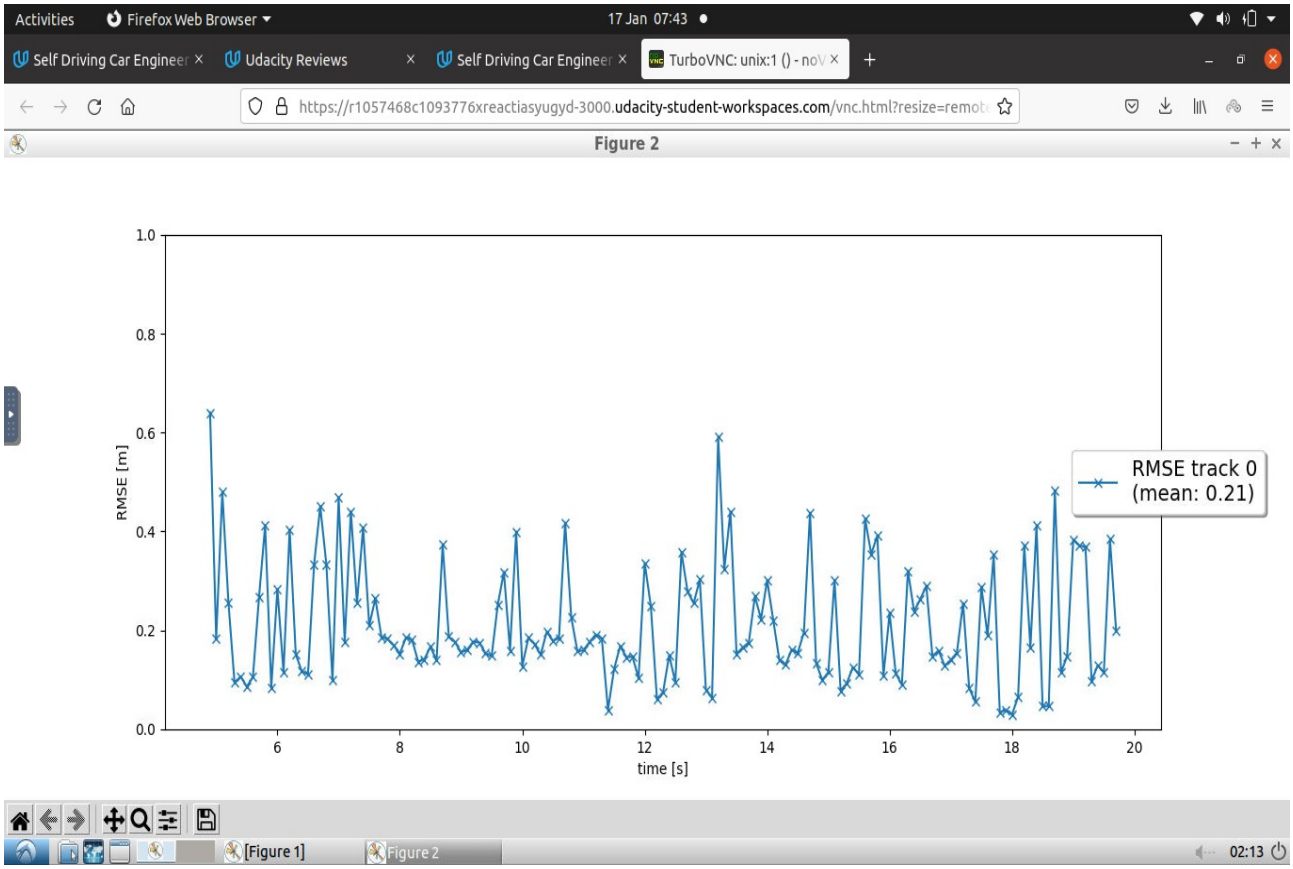
The changes are made in 'loop_over_dataset.py'

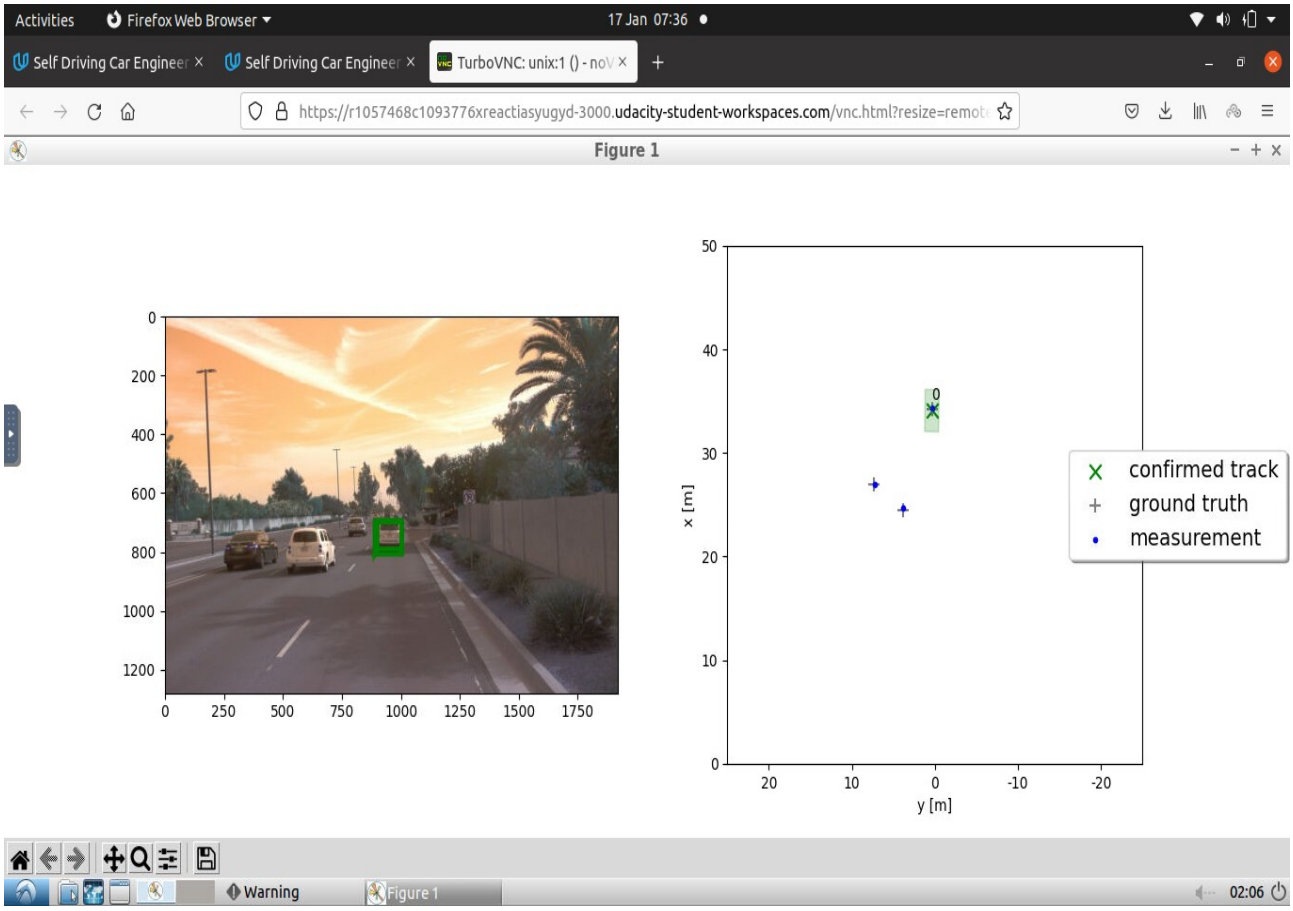
```
79
80 ## Selective execution and visualization
81 exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_
   'validate_object_labels', 'measure_detection_performance'; options not in the list will be loaded from file
82 exec_tracking = ['perform_tracking'] # options are 'perform_tracking'
83 exec_visualization = ['show_tracks'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image', 'show_objects_
   'show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_movie'
84 exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
85 vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
86
```

The changes are made in "filter.py"

```python
class Filter:
    '''Kalman filter class'''
    def __init__(self):
        self.dim_state = params.dim_state# process model dimension
        self.dt = params.dt # time increment
        self.q = params.q # process noise variable for Kalman filter Q


    def F(self):
        ############
        # TODO Step 1: implement and return system matrix F
        N = self.dim_state
        F = np.identity(self.dim_state).reshape(N,N)
        F[0, 3] = self.dt
        F[1, 4] = self.dt
        F[2, 5] = self.dt

        ############

        return np.matrix(F)

        ############
        # END student code
        ############

    def Q(self):
        ############
        # TODO Step 1: implement and return process noise covariance Q
        ############
        dt = self.dt
        q = self.q
        q1 = dt * q
        Q = np.zeros((self.dim_state, self.dim_state))
        np.fill_diagonal(Q, q1)

        return np.matrix(Q)


        ############
```

The output of EKF sample:





By seeing above figure, we have concluded fro STEP1,Tracking 3D object with a constant velocity model including height estimation, so F and Q will be 6D matrices and has implemented in above code section. This has provided RMSE graph with value of 0.21 which is less than 0.35.

Step-2: Track Management

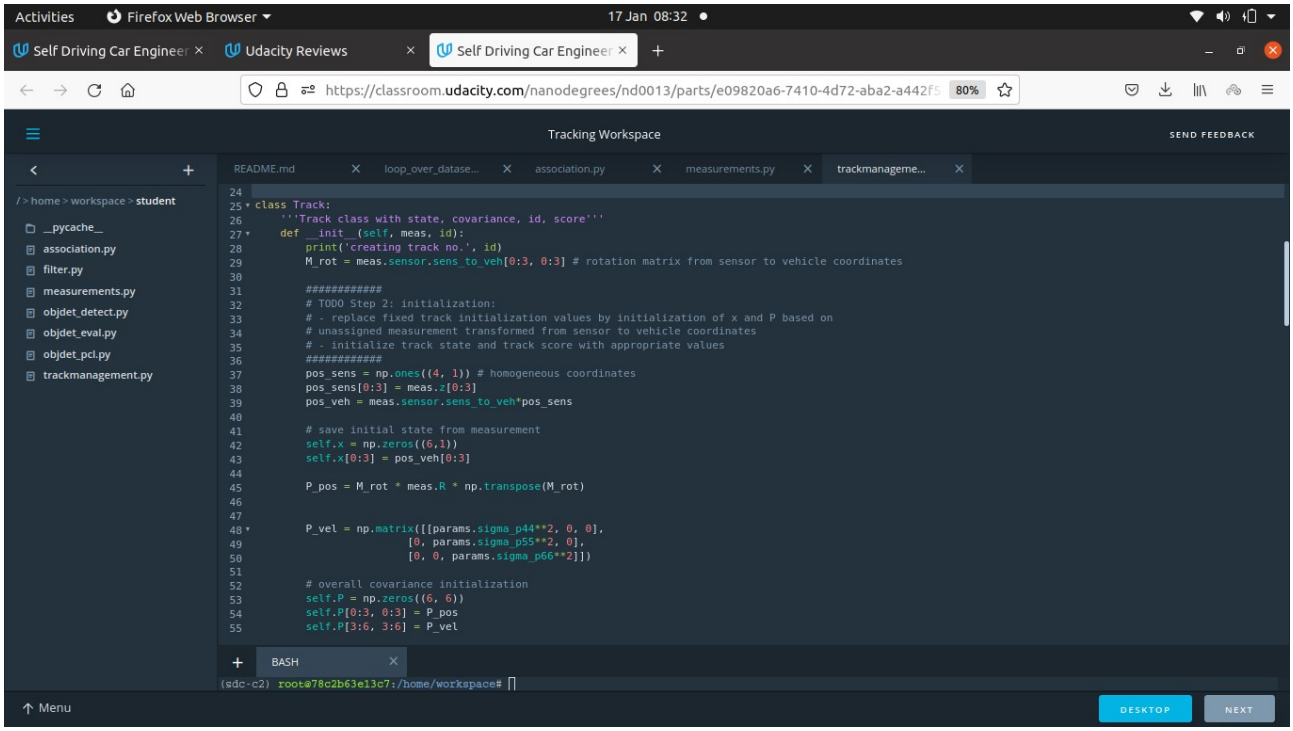
In this case, we are:

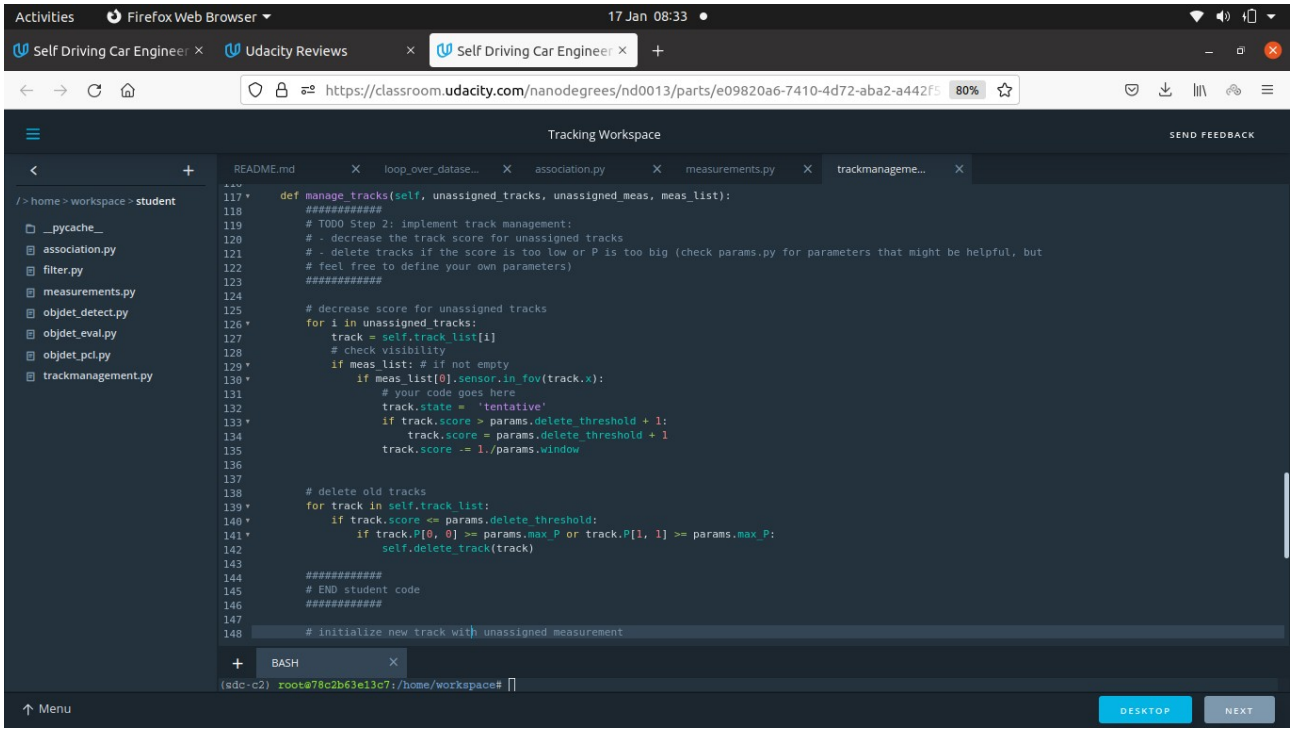
- Track initialization from unassigned measure, is implemented
- Track state and score are defined and implemented
- Old tracks are deleted for not updated tracks
- Plotation of RMSE

The changes are in the 'loop_over_dataset.py'

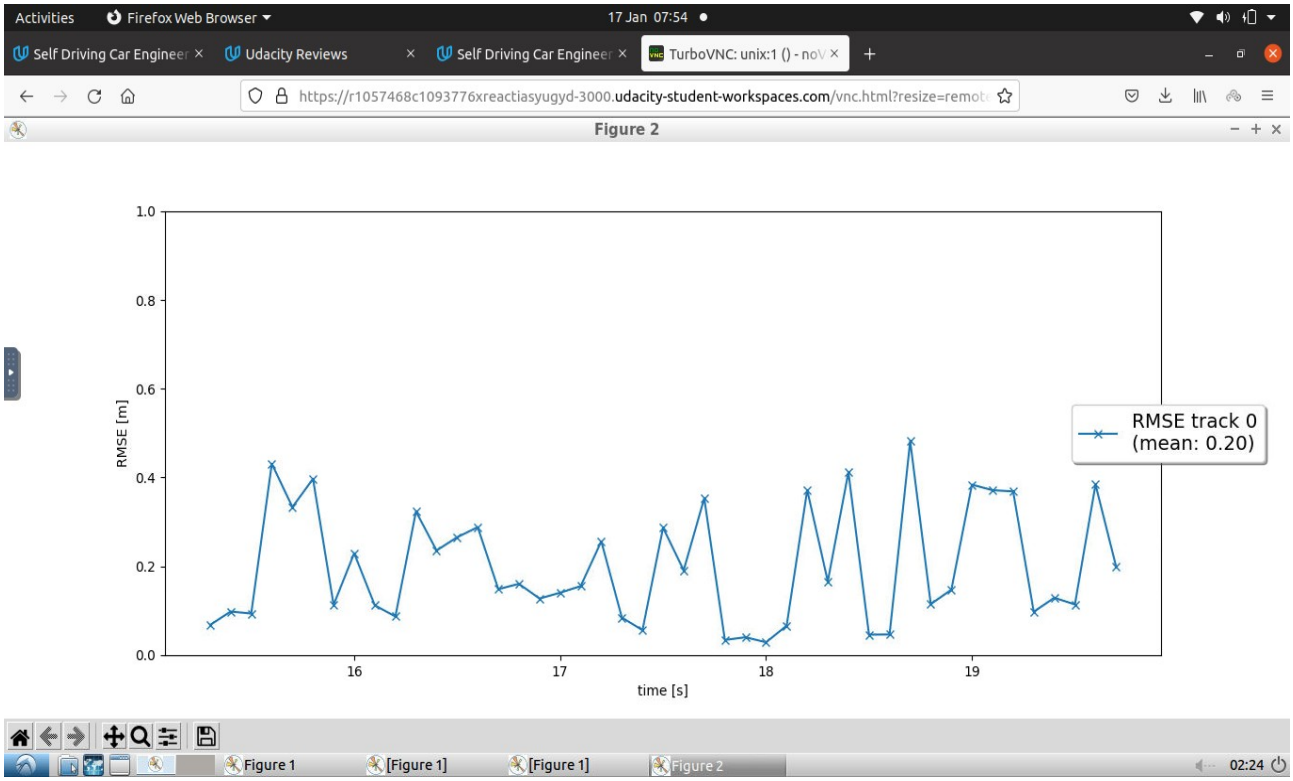
```
79
80  ## Selective execution and visualization
81  exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_
    'validate_object_labels', 'measure_detection_performance'; options not in the list will be loaded from file
82  exec_tracking = ['perform_tracking'] # options are 'perform_tracking'
83  exec_visualization = ['show_tracks'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image', 'show_objects_
    'show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_movie'
84  exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
85  vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
86
```

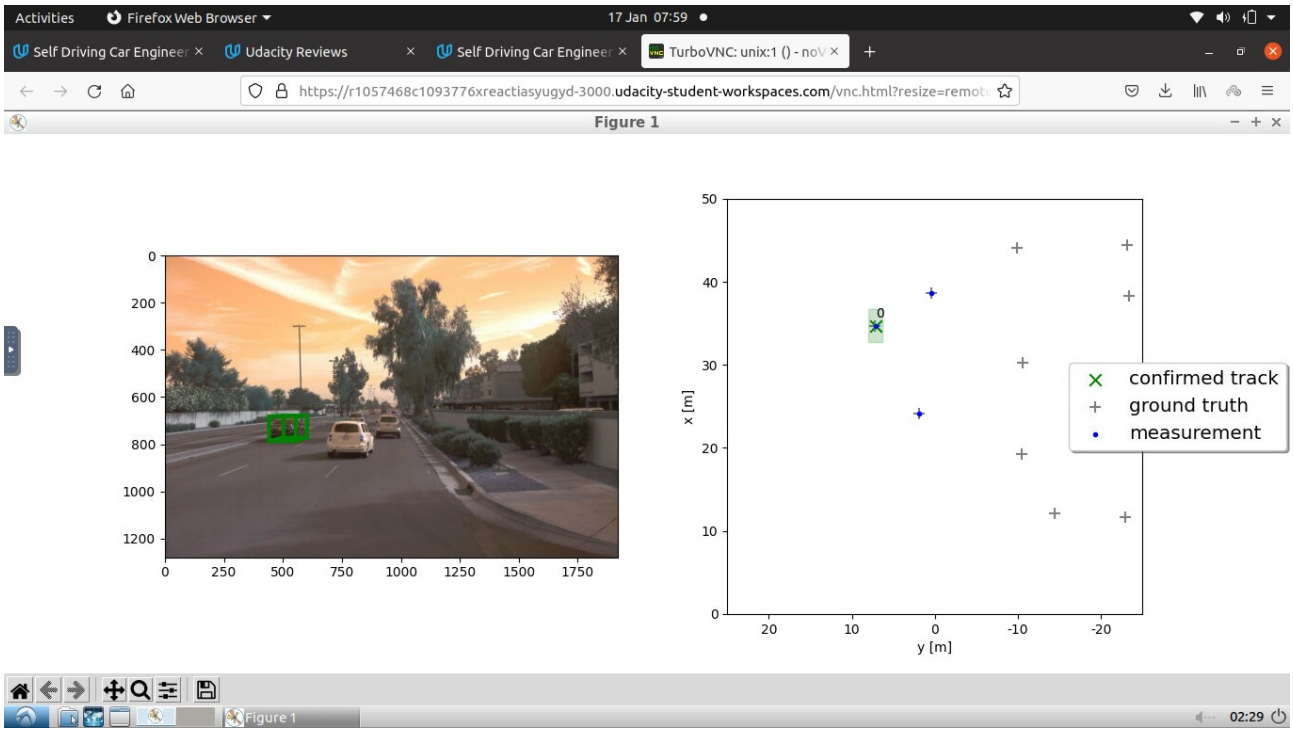
The changes are also in the "trackmanagement.py"

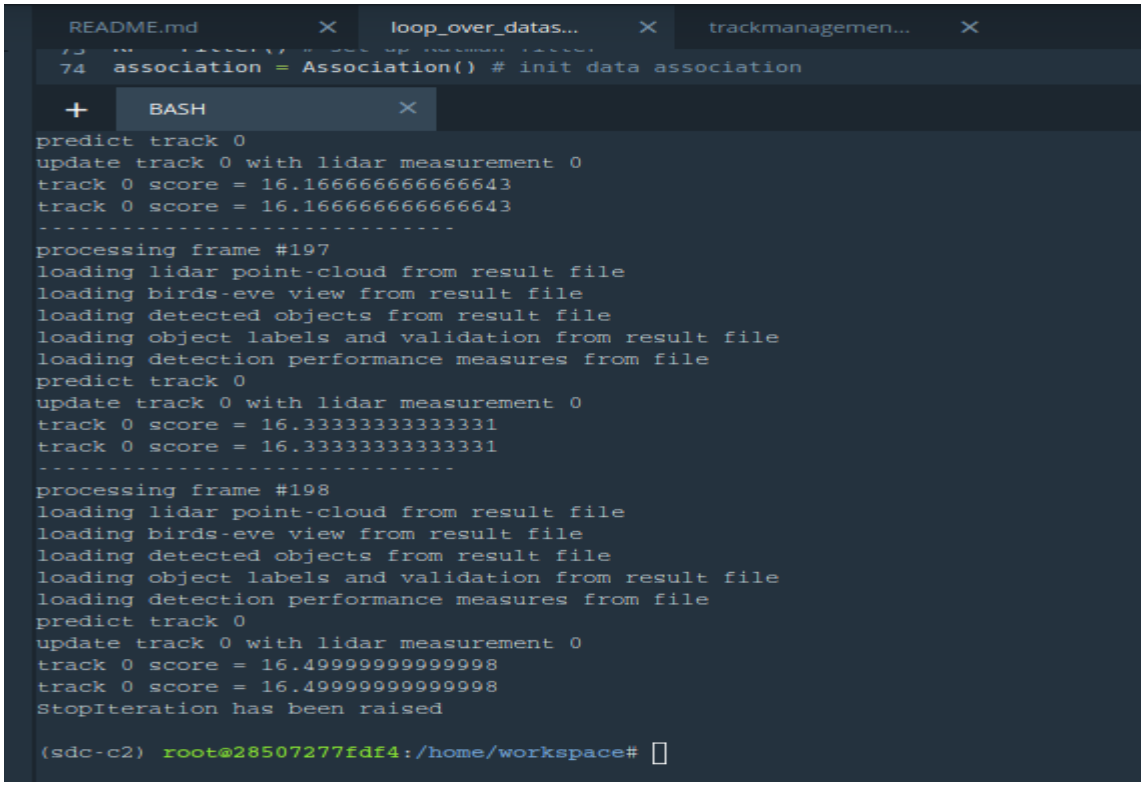




A sample preview of the Trackmanagement:







Step-3: Data Association

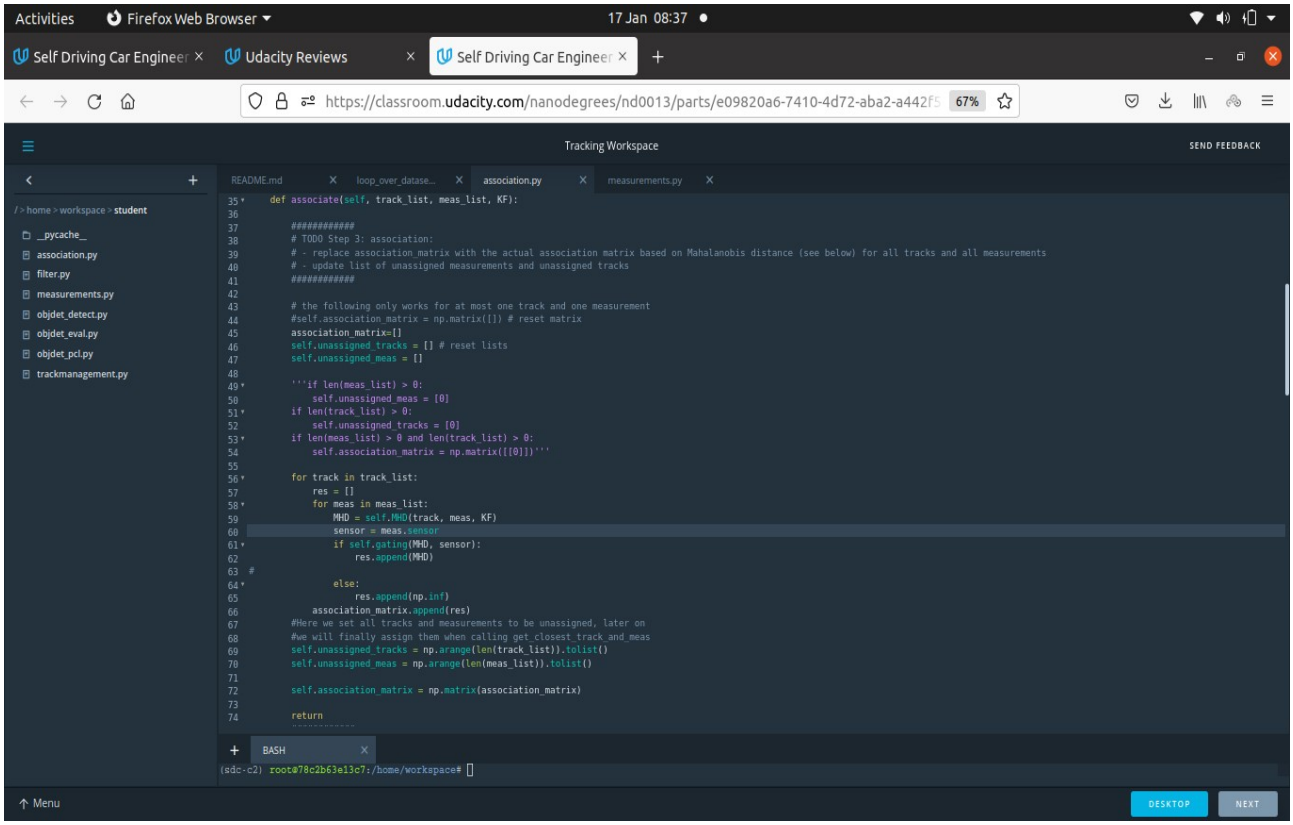
Here, particularly implementing a single nearest neighbor data association to associate measurements to tracks where output shows that each measurement is used at most once and each track is updated at most once. The visualization has shown that there are no confirmed “ghost tracks” that do not exist in reality. There may be initialized or tentative “ghost tracks” as long as they are deleted after several frames

- Nearest neighbor data association including association matrix is implemented with returns method for nearest track and measurement.
- Gating method with chi-square-distribution is implemented to reduce complexity.
- RMSE plot

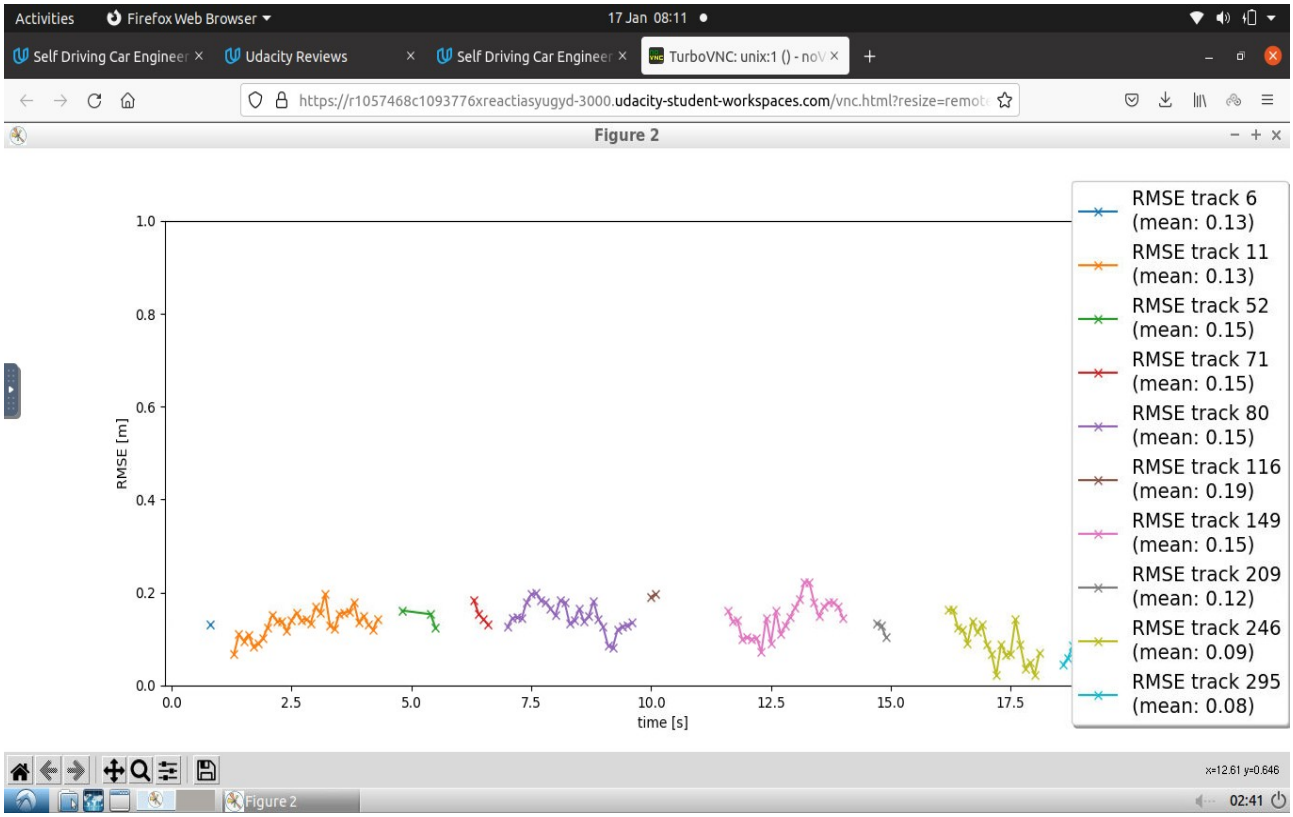
The changes are in "loop_over_dataset.py"

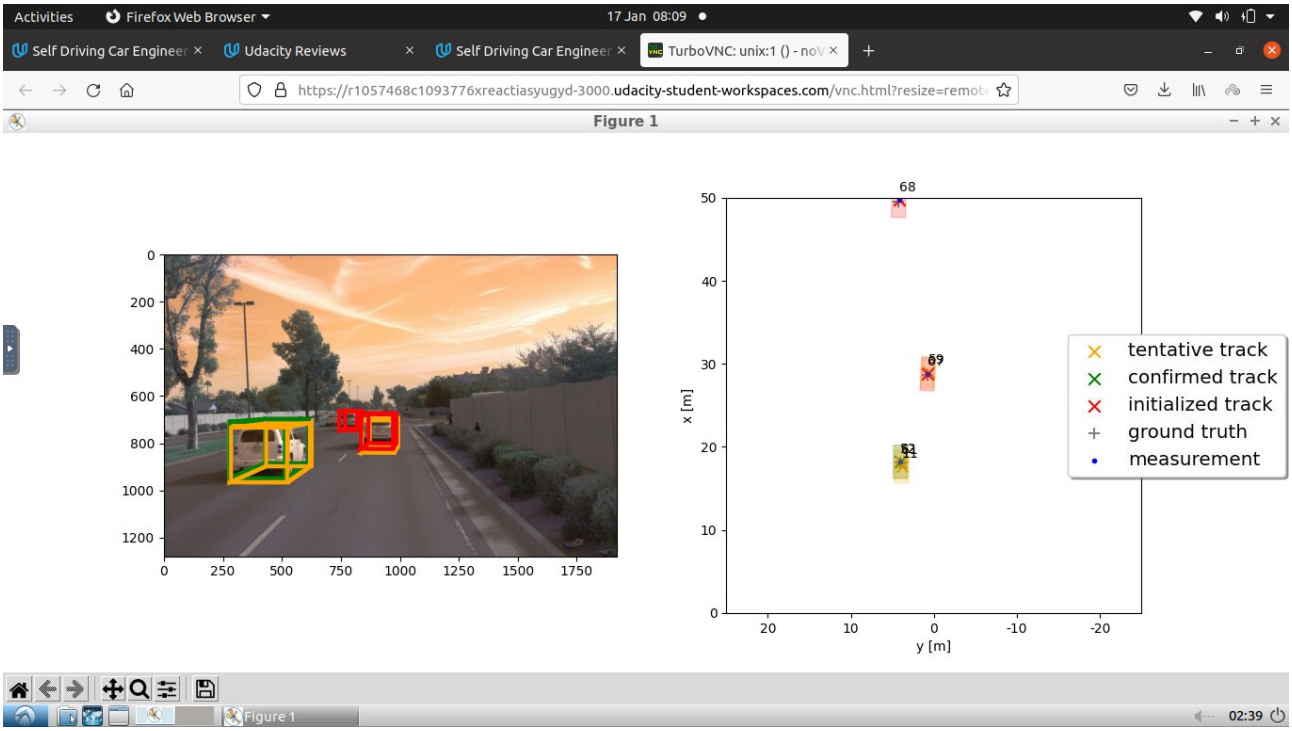
```
79
80  ## Selective execution and visualization
81  exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_
    'validate_object_labels', 'measure_detection_performance'; options not in the list will be loaded from file
82  exec_tracking = ['perform_tracking'] # options are 'perform_tracking'
83  exec_visualization = ['show_tracks'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image', 'show_objects_
    'show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_movie'
84  exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
85  vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
86
```

The changes are inside the "association.py" file:

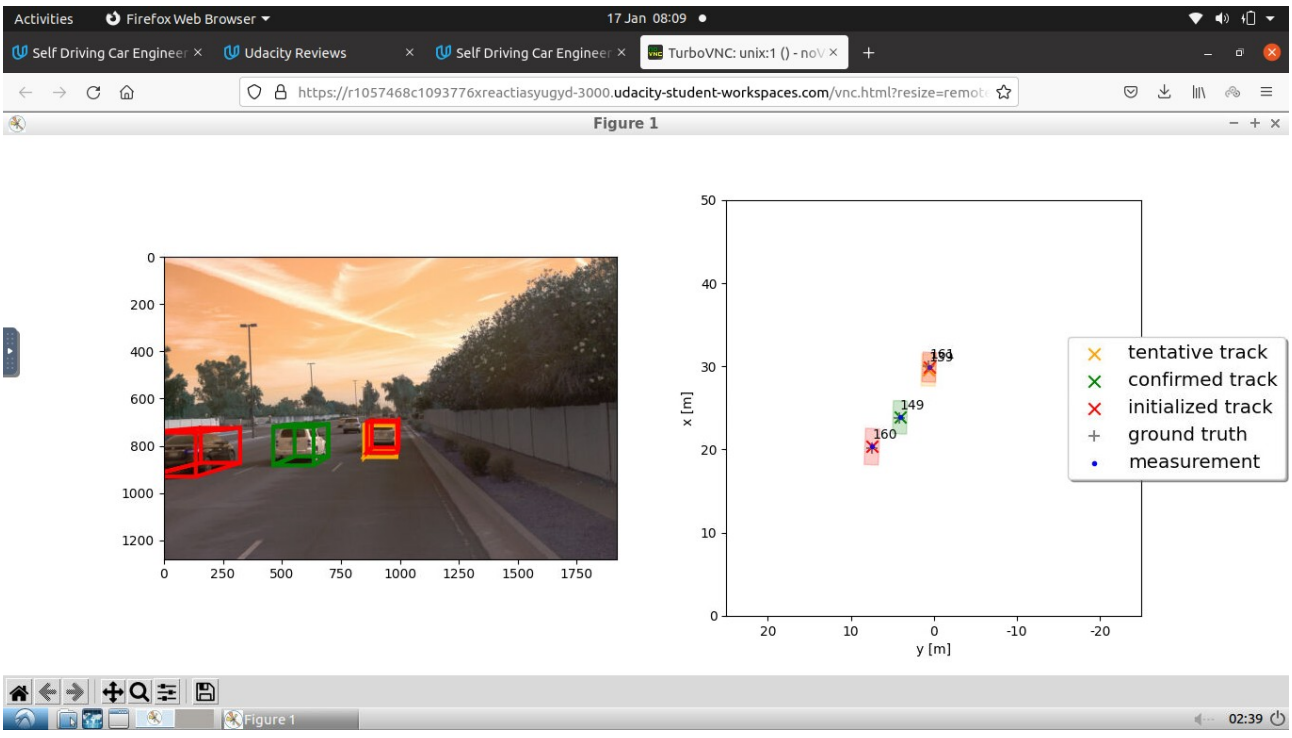


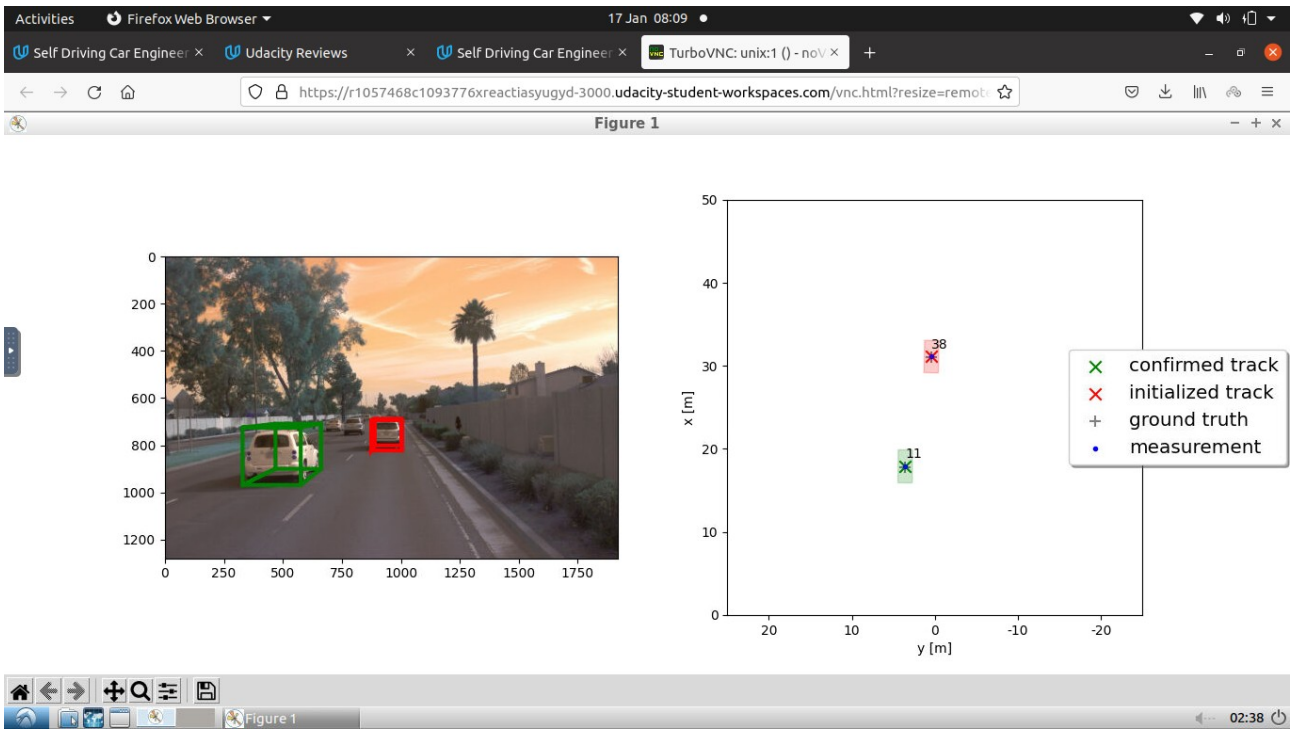
A sample preview of the association measurement:

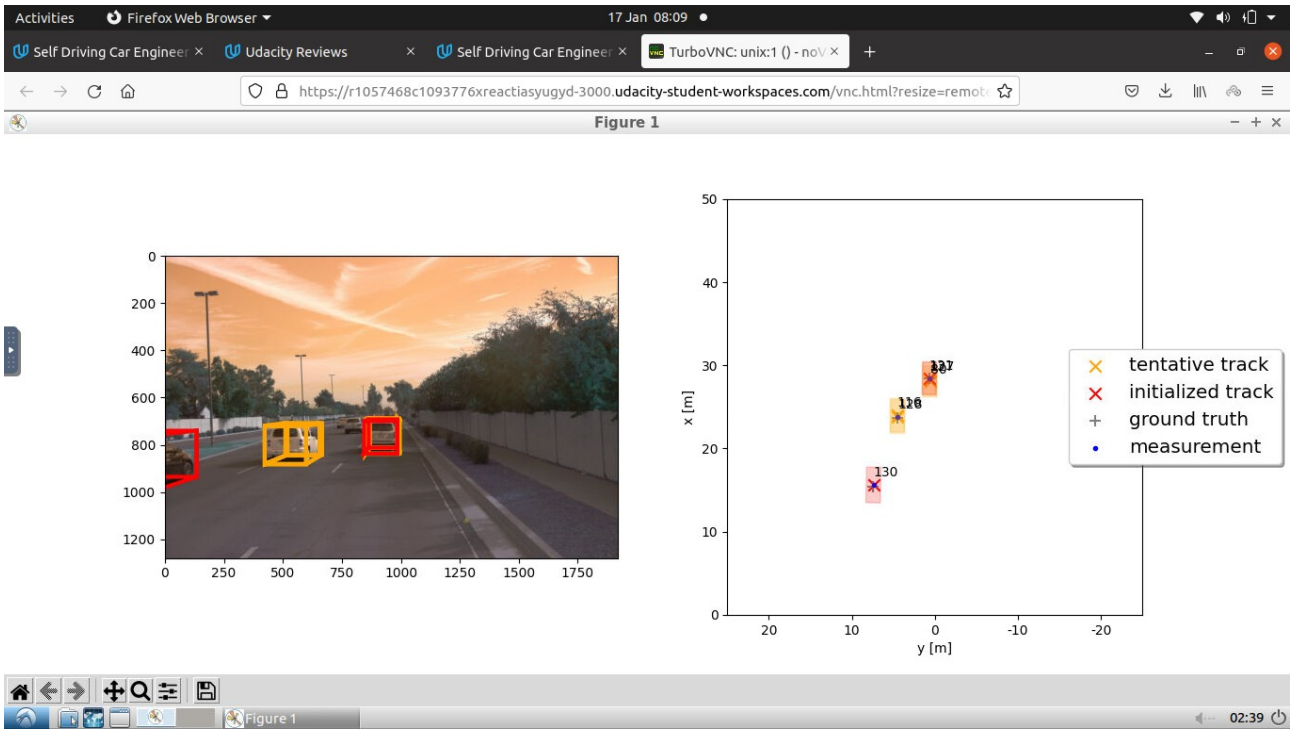


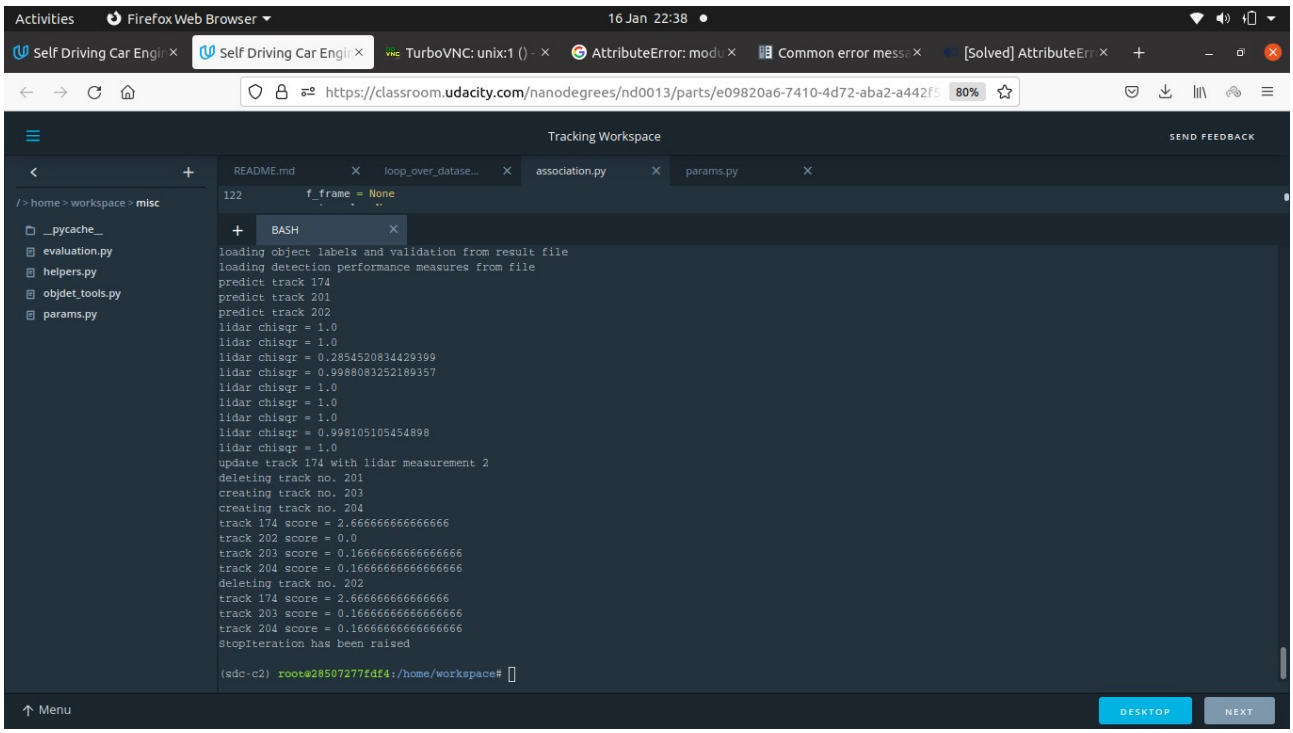



Activities
Firefox Web Browser
17 Jan 08:09
Self Driving Car Engineer
Udacity Reviews
Self Driving Car Engineer
TurboVNC: unix:1 () - noV
https://r1057468c1093776xreactiasyugyd-3000.udacity-student-workspaces.com/vnc.html?resize=remot
Figure 1
x [m]
y [m]
tentative track
confirmed track
initialized track
ground truth
measurement
78
71
Figure 1
02:39


Activities
Firefox Web Browser
17 Jan 08:09
Self Driving Car Engineer
Udacity Reviews
Self Driving Car Engineer
TurboVNC: unix:1 () - noV
https://r1057468c1093776xreactiasyugyd-3000.udacity-student-workspaces.com/vnc.html?resize=remot
Figure 1
x [m]
y [m]
tentative track
confirmed track
initialized track
ground truth
measurement
161
149
160
Figure 1
02:39


Activities
Firefox Web Browser
17 Jan 08:09
Self Driving Car Engineer
Udacity Reviews
Self Driving Car Engineer
TurboVNC: unix:1 () - noV
https://r1057468c1093776xreactiasyugyd-3000.udacity-student-workspaces.com/vnc.html?resize=remot
Figure 1
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750
50
40
30
20
10
0
x [m]
20
10
0
-10
-20
y [m]
38
11
confirmed track
initialized track
ground truth
measurement
Figure 1
02:38


Activities
Firefox Web Browser
17 Jan 08:09
Self Driving Car Engineer
Udacity Reviews
Self Driving Car Engineer
TurboVNC: unix:1 () - noV
https://r1057468c1093776xreactiasyugyd-3000.udacity-student-workspaces.com/vnc.html?resize=remot
Figure 1
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750
50
40
30
20
10
0
x [m]
20
10
0
-10
-20
y [m]
130
tentative track
initialized track
ground truth
measurement
Figure 1
02:39

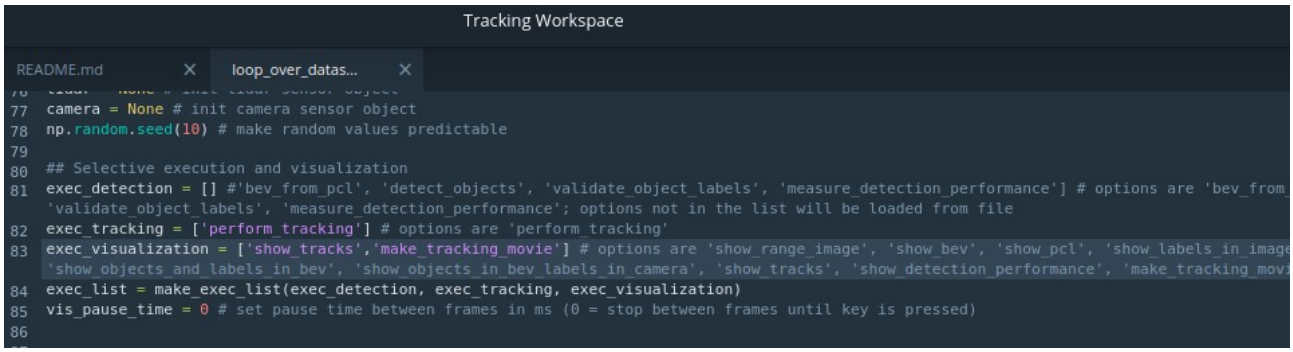


Step-4: Sensor Fusion

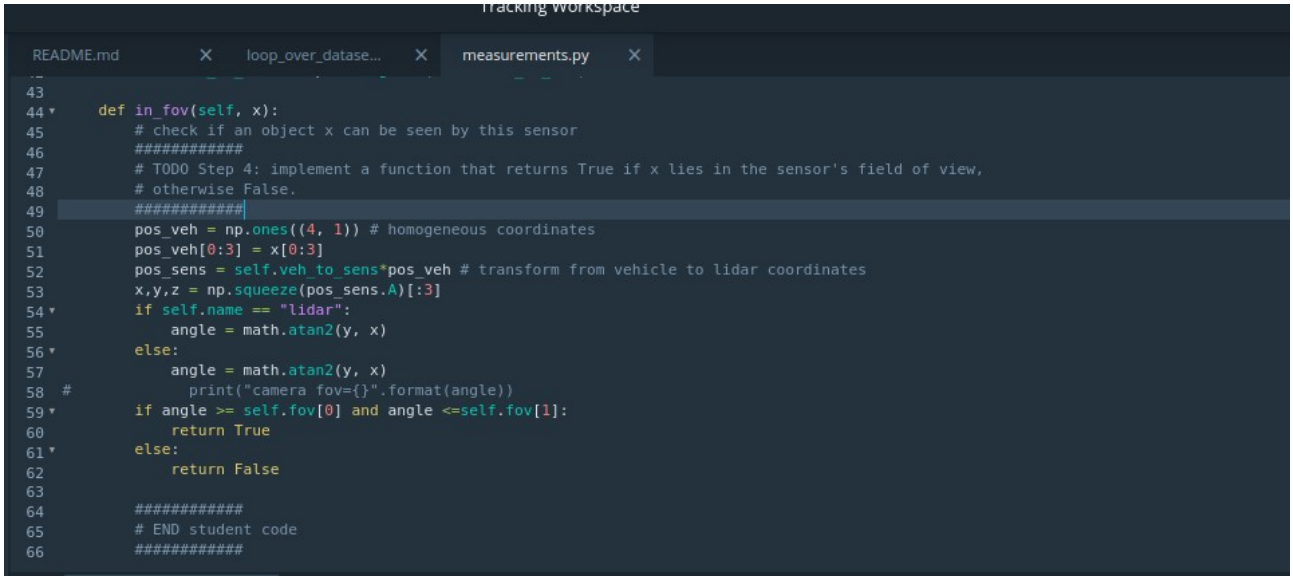
In this step, implementation the nonlinear camera measurement model and then finally combining the sensor fusion module for completing camera-lidar fusion where output shows lidar updates followed by camera updates.

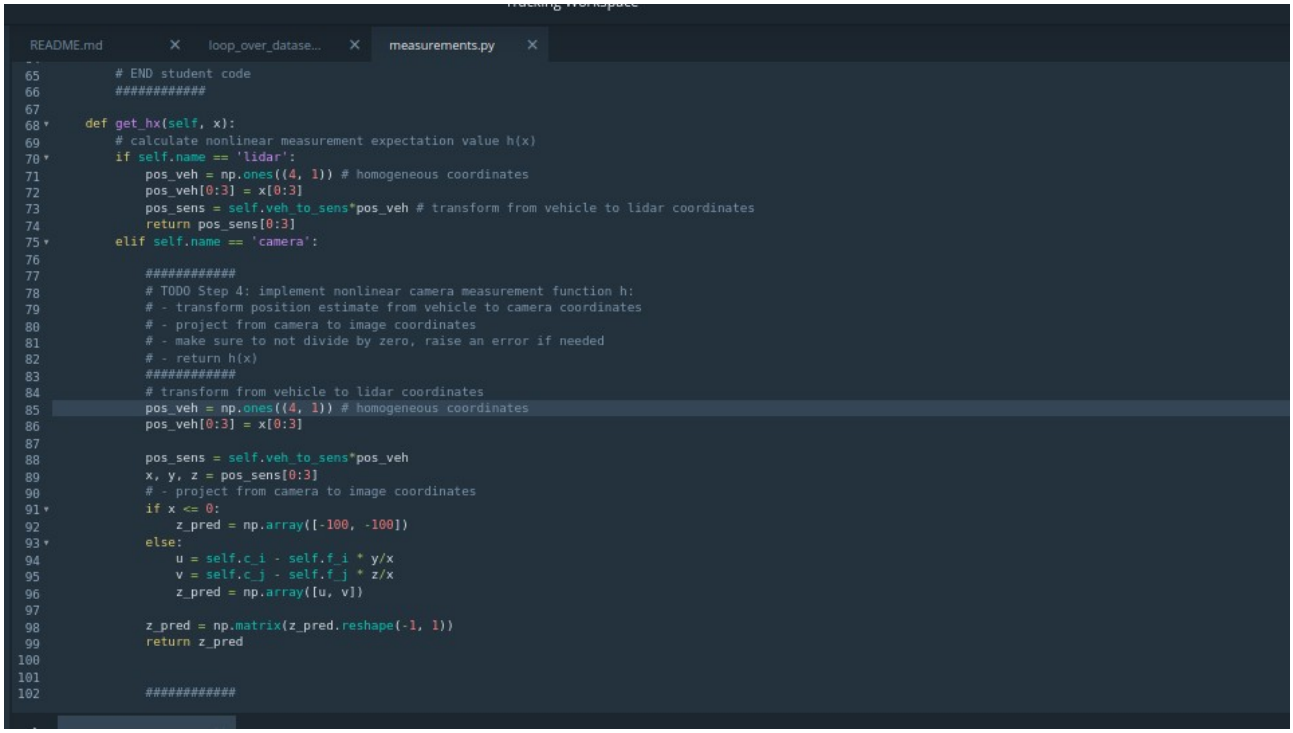
- Camera measurements including appropriate covariance matrix R are implemented.
- Non linear camera measurement model h(x) is implemented.
- A method checked whether an object in range of the camera or outside the field of view is implemented.
- Produce output video using flags ‘make-track_movies’

The changes in the code are:

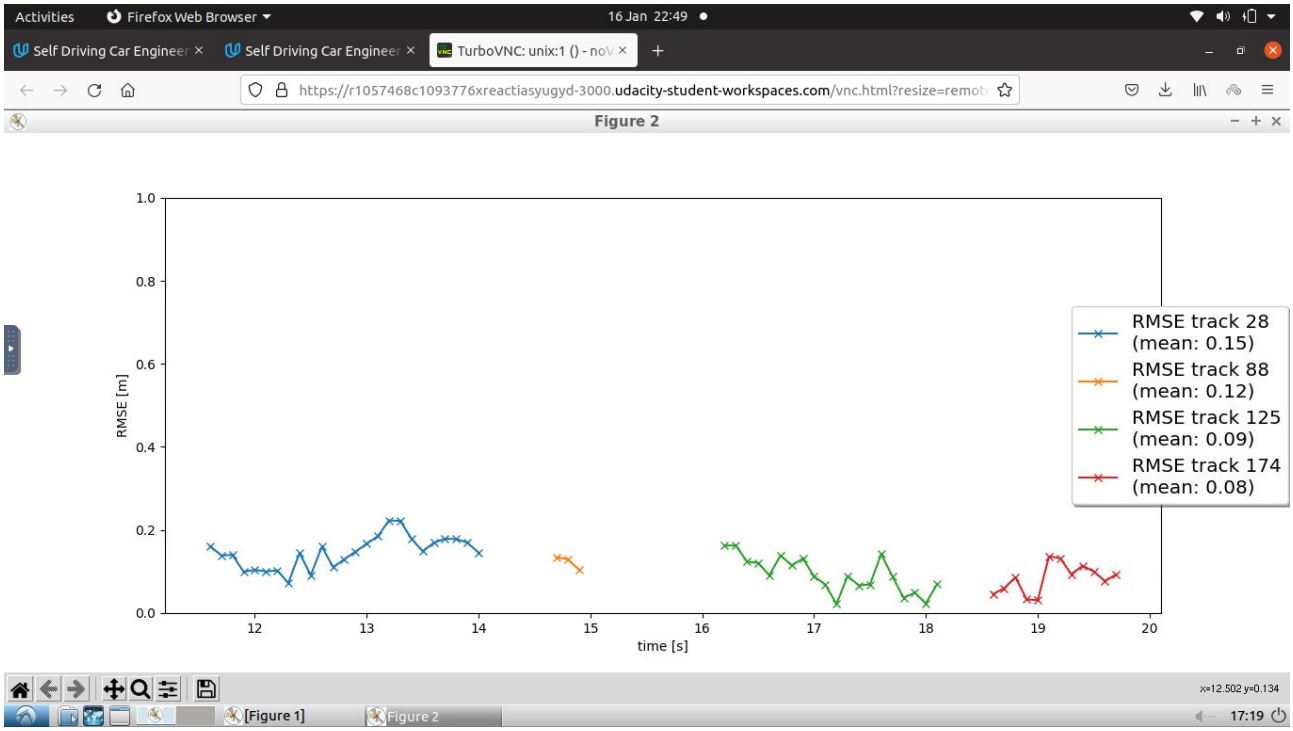


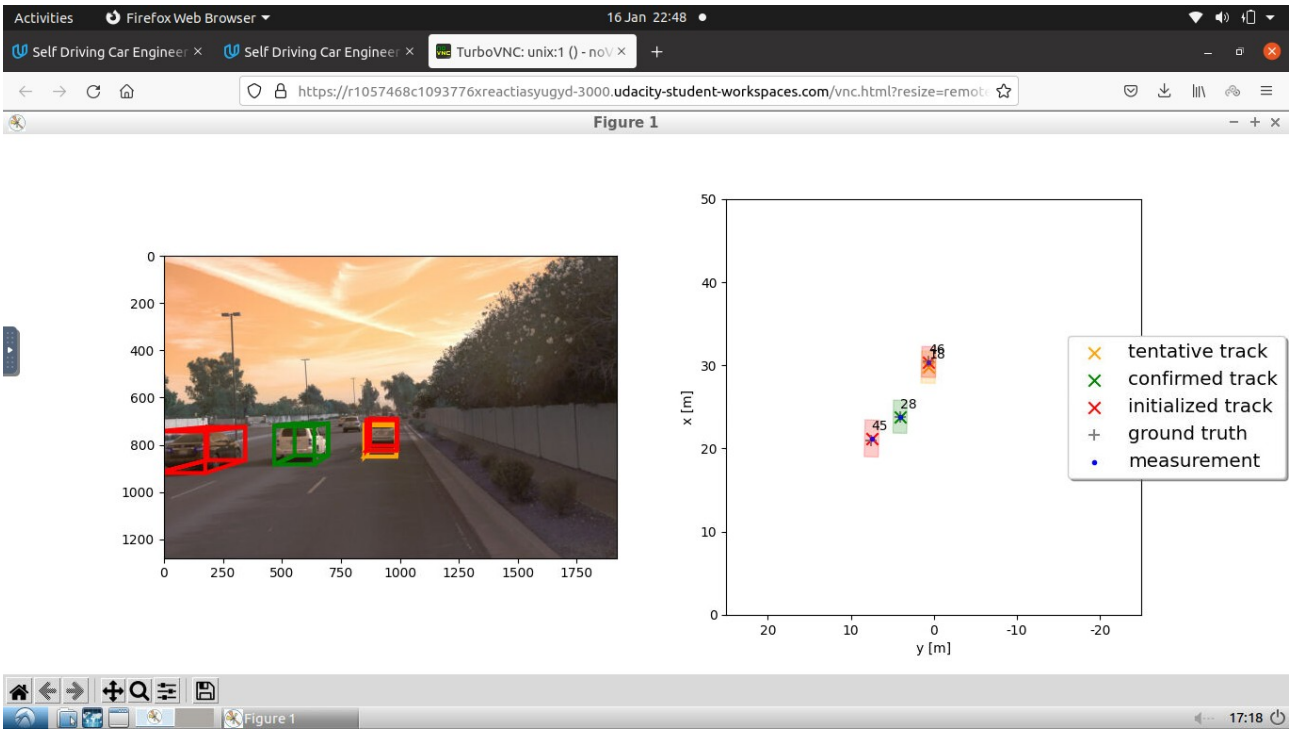
The changes for "measurements.py" :

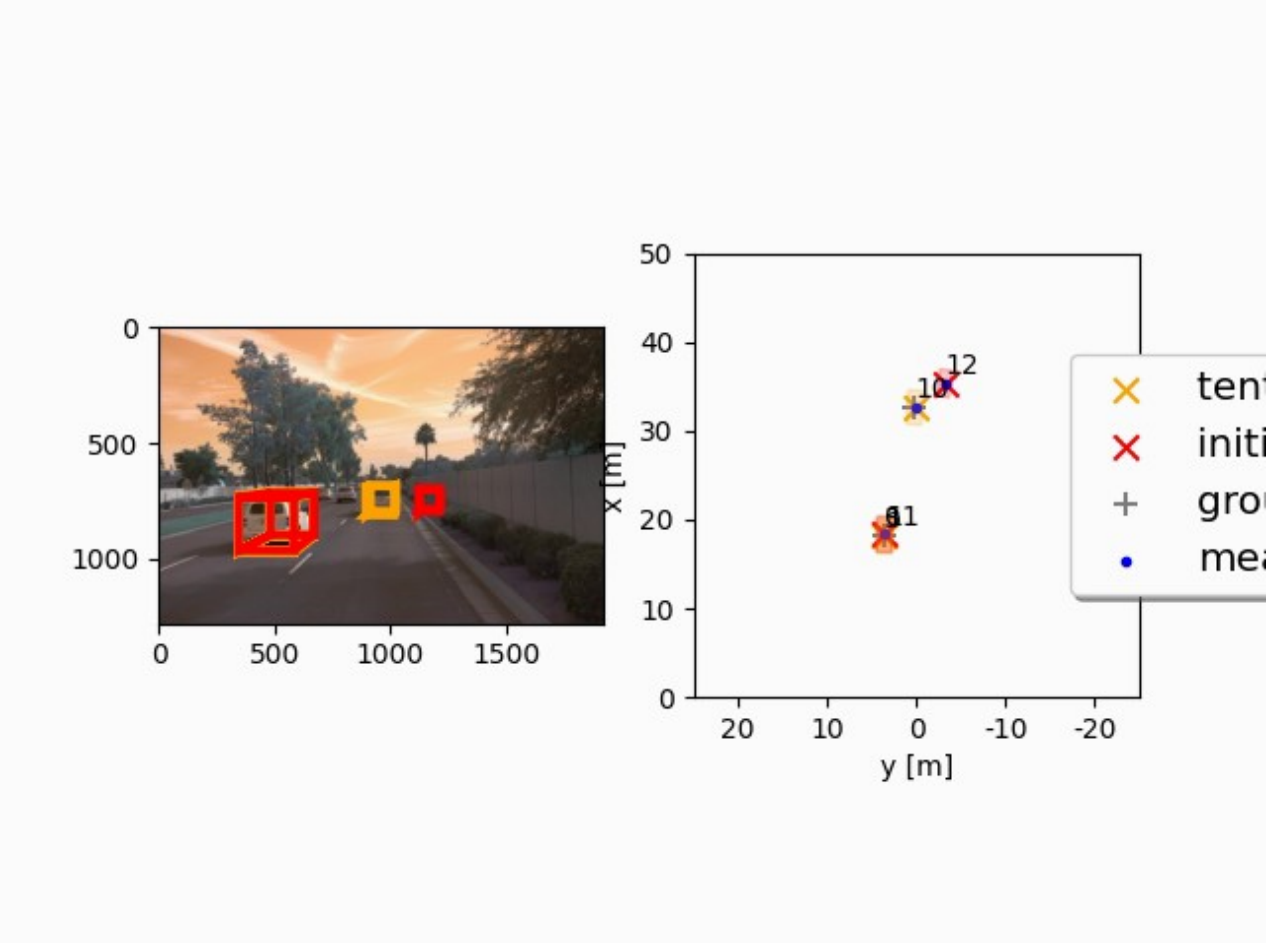




Results of Measurements.py:







This is video file(click on this image, go to left bottom corner,play button option will seen.)

Summary of Sensor Fusion and Object Tracking

From the project, it is understandable that for a good multi-tracking, lidar and camera should be used . In order to make more efficient these sensor, Extended kalman Filter is necessary for multi tracking under their field view because this algorithm provide more certainty behaviour of having next object around it by updating and predicting future tracks of position and velocity.Along this ,optimization is done through Root Mean Sqaure Error(RMSE) which is equivalent to 0.35 or less than it for better performance and we got 0.21 RMSE value.