

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

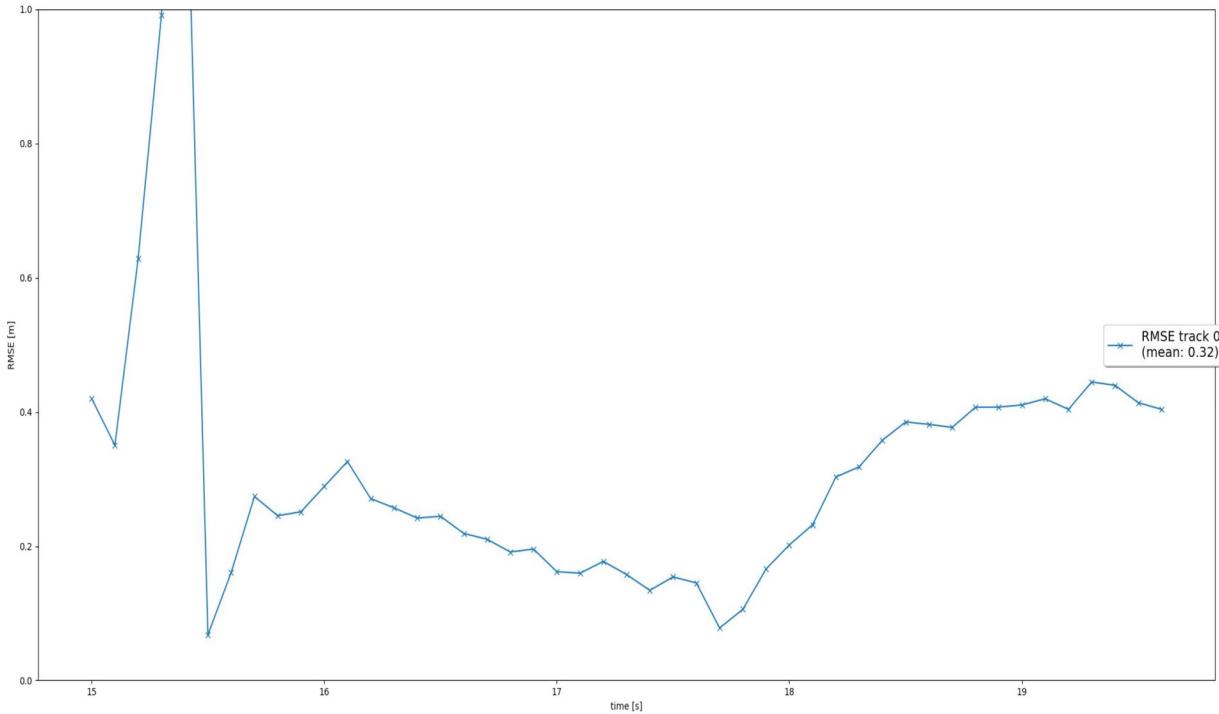
The changes are made in 'loop_over_dataset.py'

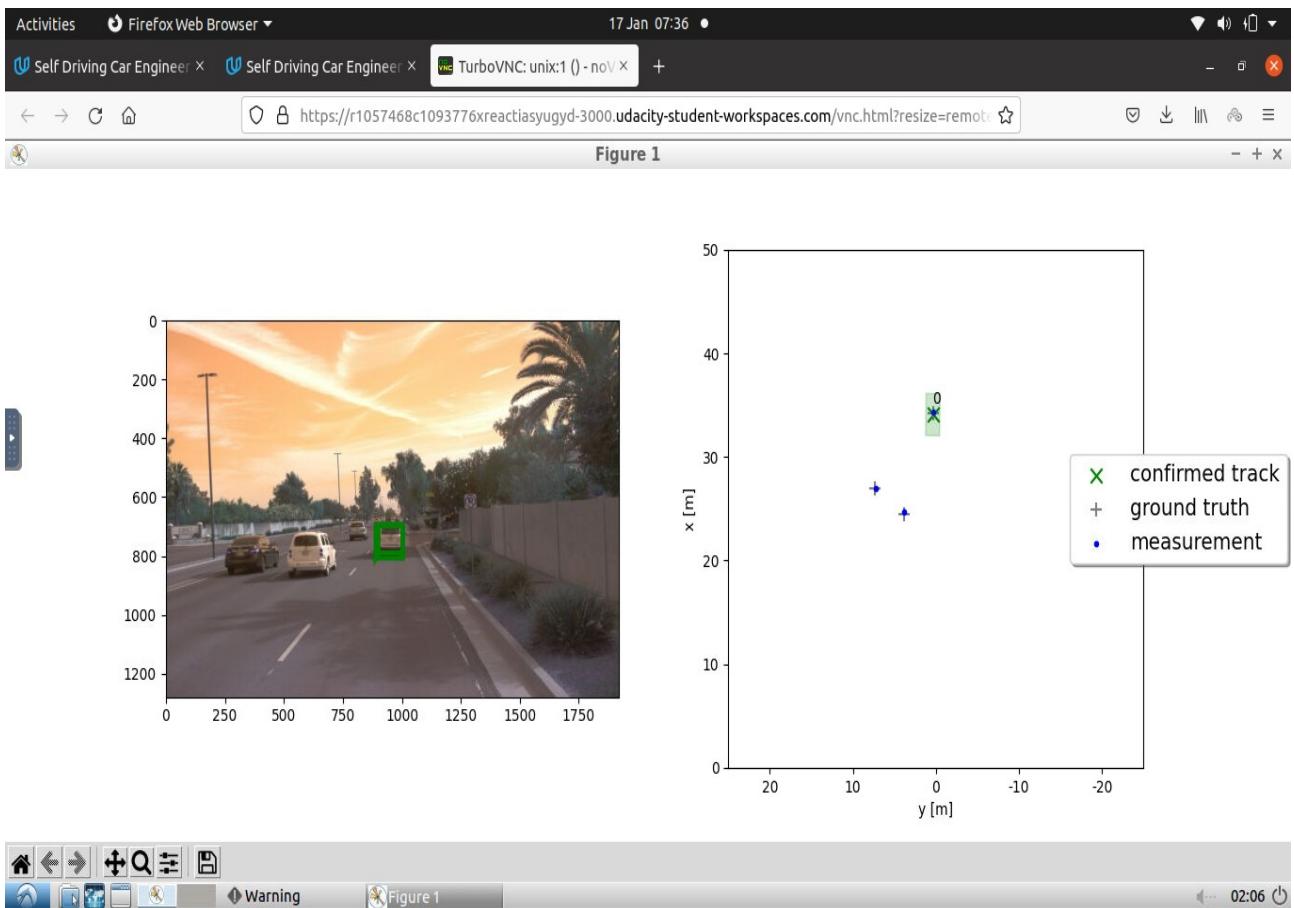
```
79 ## Selective execution and visualization
80 exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_
81 'validate_object_labels', 'measure_detection_performance'; options not in the list will be loaded from file
82 exec_tracking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

The changes are made in "filter.py"

```
Activities Firefox Web Browser ▾ 19 Jan 22:11
Self Driving Car × Udacity Reviews × Udacity Reviews × Self Driving Car × Self Driving Car × Knowledge - Uda × New Tab × + - ×
Tracking Workspace
filter.py
38
39
40
41
42
43
44
45
46
47
48
49
50+
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68+
69
70
# PyCharm Step 1: implement and return system matrix F
N = self.dim_state
F[0, 3] = self.dt
F[1, 4] = self.dt
F[2, 5] = self.dt
return np.matrix(F)
# PyCharm Step 1: implement and return process noise covariance Q
q = params.q
dt = params.dt
q3 = (q/3)*dt**3
q2 = (q/2)*dt**2
q1 = q*dt
return np.matrix([[q3, 0, 0, q2, 0, 0],
[0, q3, 0, 0, q2, 0],
[0, 0, q3, 0, 0, q2],
[q2, 0, 0, q1, 0, 0],
[0, q2, 0, 0, q1, 0],
[0, 0, q2, 0, 0, q1]])
# END student code
#####
# PyCharm Step 1: implement and return system matrix F
N = self.dim_state
F[0, 3] = self.dt
F[1, 4] = self.dt
F[2, 5] = self.dt
return np.matrix(F)
# PyCharm Step 1: implement and return process noise covariance Q
q = params.q
dt = params.dt
q3 = (q/3)*dt**3
q2 = (q/2)*dt**2
q1 = q*dt
return np.matrix([[q3, 0, 0, q2, 0, 0],
[0, q3, 0, 0, q2, 0],
[0, 0, q3, 0, 0, q2],
[q2, 0, 0, q1, 0, 0],
[0, q2, 0, 0, q1, 0],
[0, 0, q2, 0, 0, q1]])
# END student code
#####
+ BASH ×
(rainbow) root@80a867fcf4d7:/home/workspace#
```

The output of EKF sample(step1):





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'

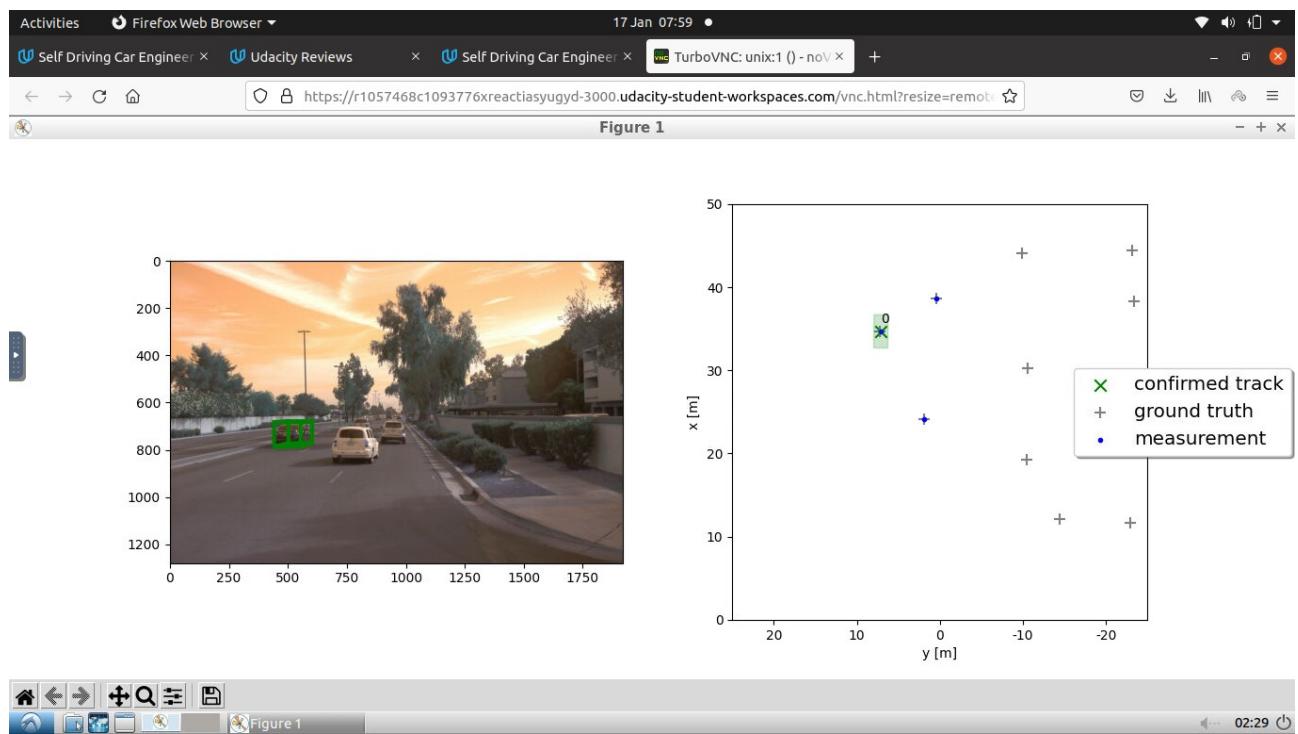
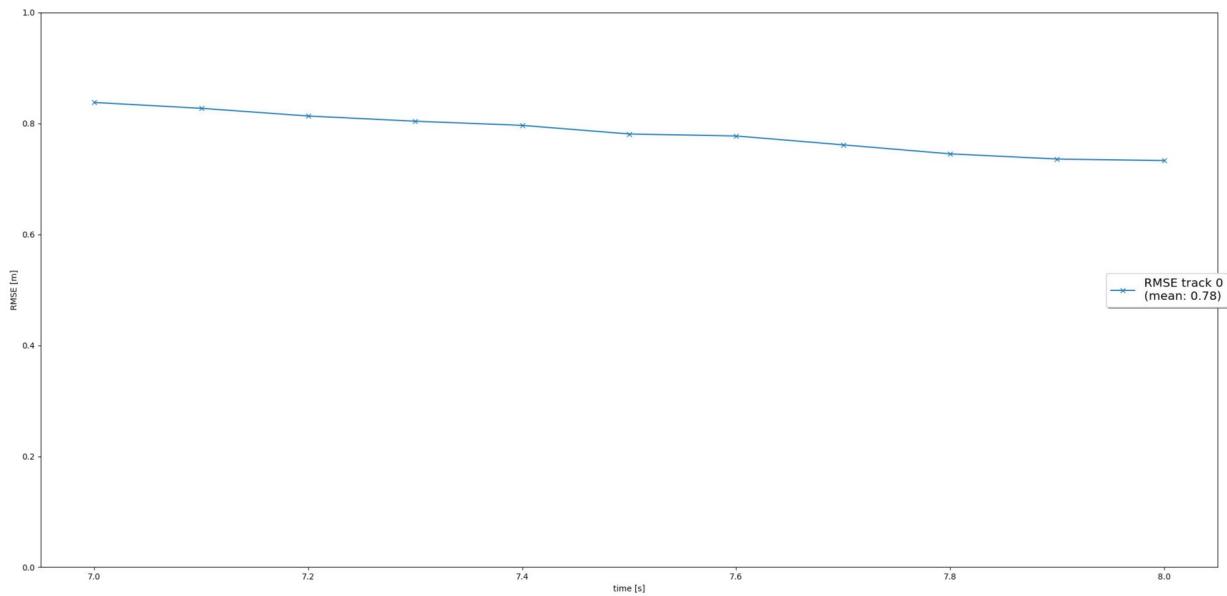
```
79 ## Selective execution and visualization
80 exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_
81 'validate_object_labels', 'measure_detection_performance'; options not in the list will be loaded from file
82 exec_tracking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

The changes are also in the "trackmanagement.py"

```
25 * class Track:
26     '''Track class with state, covariance, id, score'''
27 *     def __init__(self, meas, id):
28         print('creating track no.', id)
29         M_rot = meas.sensor.sens_to_veh[0:3, 0:3] # rotation matrix from sensor to vehicle coordinates
30
31         #####
32         # TODO Step 2: initialization:
33         # - replace fixed track initialization values by initialization of x and P based on
34         # unassigned measurement transformed from sensor to vehicle coordinates
35         # - initialize track state and track score with appropriate values
36         #####
37         M_sens2veh = meas.sensor.sens_to_veh
38         homog_z = np.vstack([meas.z, 1])
39         pos_veh = M_sens2veh * homog_z
40         # save initial state from measurement
41         self.x = np.zeros((6,1))
42         self.x[0:3] = pos_veh[0:3]
43
44         P_pos = M_rot * meas.R * np.transpose(M_rot)
45
46
47         P_vel = np.matrix([[params.sigma_p44**2, 0, 0],
48                            [0, params.sigma_p55**2, 0],
49                            [0, 0, params.sigma_p66**2]])
50
51         # overall covariance initialization
52         self.P = np.zeros((6, 6))
53         self.P[0:3, 0:3] = P_pos
54         self.P[3:6, 3:6] = P_vel
55
56         self.state = 'initialized'
57         self.score = 1./params.window
```

```
115     def manage_tracks(self, unassigned_tracks, unassigned_meas, meas_list):
116         #####
117         # TODO Step 2: implement track management:
118         # - decrease the track score for unassigned tracks
119         # - delete tracks if the score is too low or P is too big (check params.py for parameters that might be helpful, but
120         # feel free to define your own parameters)
121         #####
122
123         # decrease score for unassigned tracks
124         for i in unassigned_tracks:
125             track = self.track_list[i]
126             # check visibility
127             if meas_list: # if not empty
128                 if meas_list[0].sensor.in_fov(track.x):
129                     # your code goes here
130                     track.score -= 1./params.window
131                     #pass
132
133         # delete old tracks
134         for track in self.track_list:
135             x_pred.variance = track.P[0,0]
136             y_pred.variance = track.P[1,1]
137             if (track.score <= params.delete_threshold and track.state == 'confirmed') or x_pred.variance>params.max_P or y_pred.variance>params.max_P:
138                 self.delete_track(track)
139
140         #####
141         # END student code
142         #####
143
144
145         # initialize new track with unassigned measurement
146         for j in unassigned_meas:
```

A sample preview of the Trackmanagement:



```
README.md      X  loop_over_datas...      X  trackmanagemen...      X
74 association = Association() # init data association
+
+ BASH          X
predict track 0
update track 0 with lidar measurement 0
track 0 score = 16.16666666666643
track 0 score = 16.16666666666643
-----
processing frame #197
loading lidar point-cloud from result file
loading birds-eye view from result file
loading detected objects from result file
loading object labels and validation from result file
loading detection performance measures from file
predict track 0
update track 0 with lidar measurement 0
track 0 score = 16.33333333333331
track 0 score = 16.33333333333331
-----
processing frame #198
loading lidar point-cloud from result file
loading birds-eye view from result file
loading detected objects from result file
loading object labels and validation from result file
loading detection performance measures from file
predict track 0
update track 0 with lidar measurement 0
track 0 score = 16.49999999999998
track 0 score = 16.49999999999998
StopIteration has been raised
(sdc-c2) root@28507277fdf4:/home/workspace#
```

Step-3: Data Association

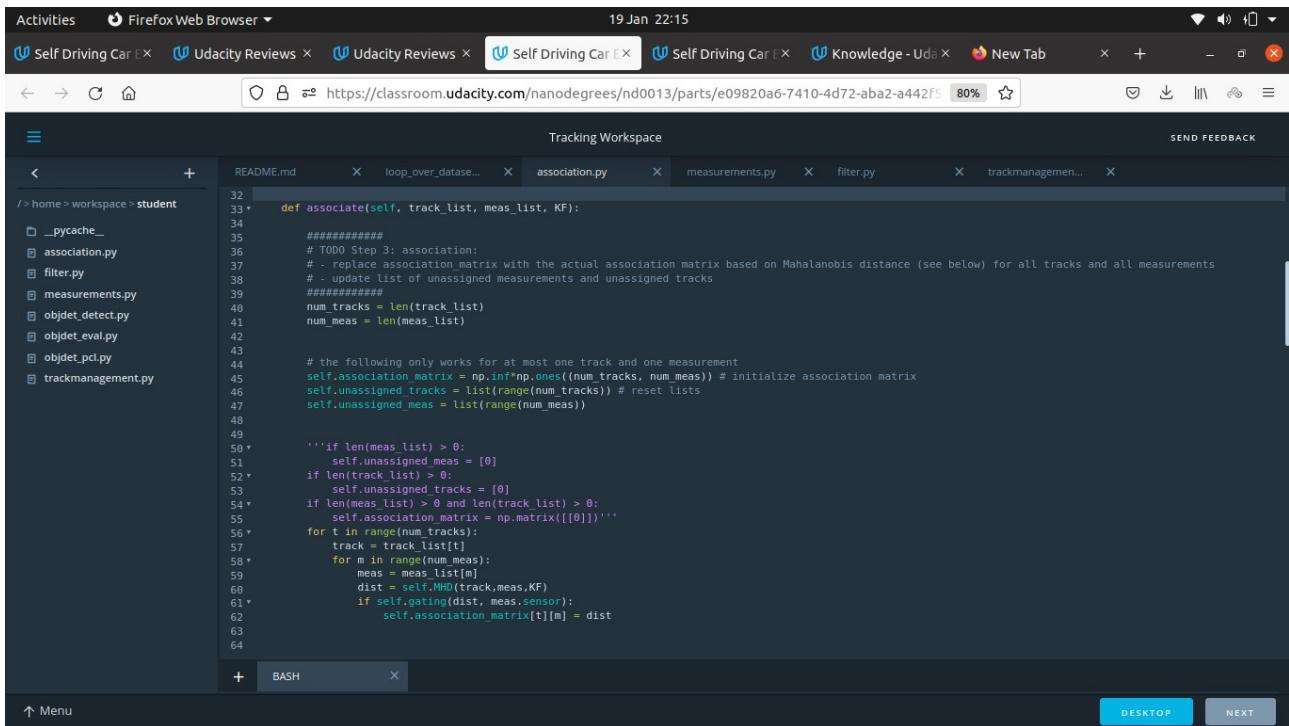
Here, particularly implementing a single nearest neighbor data association to associate measurements to tracks where output shows that each measurement is used at most once and each track is updated at most once. The visualization has shown that there are no confirmed “ghost tracks” that do not exist 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 ## Selective execution and visualization
80 exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_
81 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_
84 show_objects_in_bev_labels_in_camera', 'show_tracks', 'show_detection_performance', 'make_tracking_movie'
85 exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
86 vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)
87
```

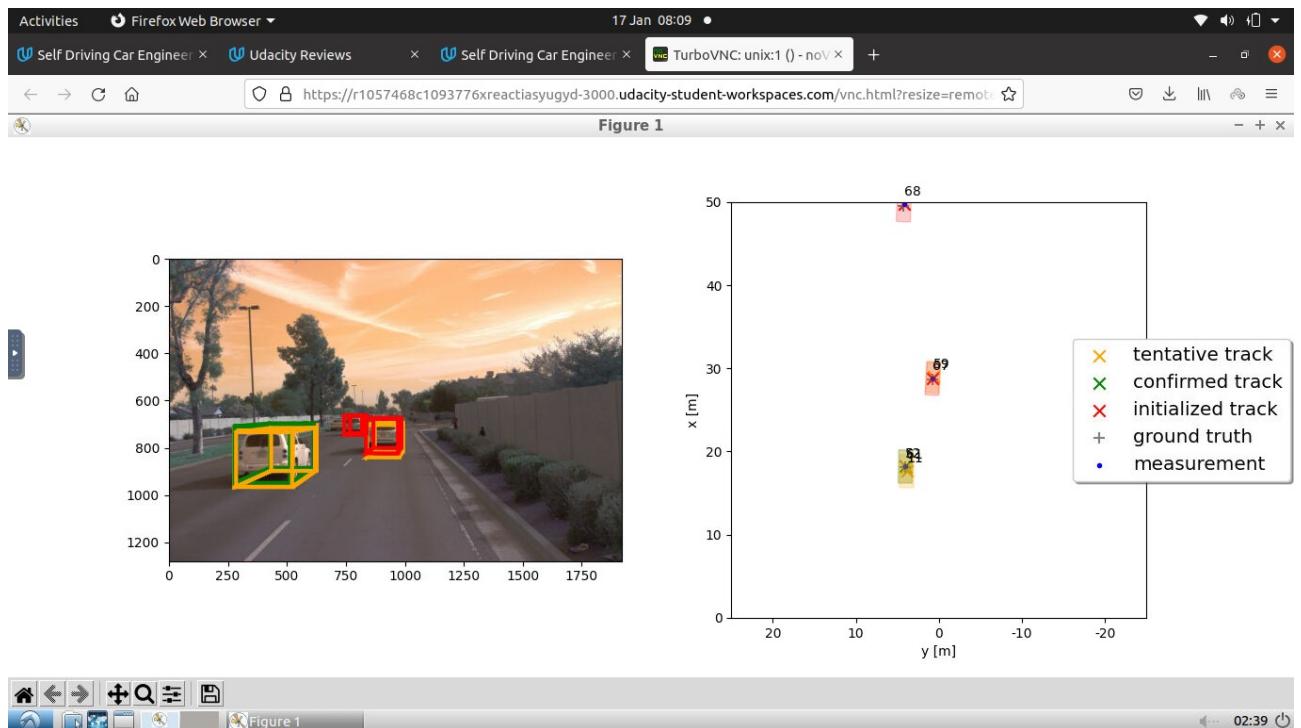
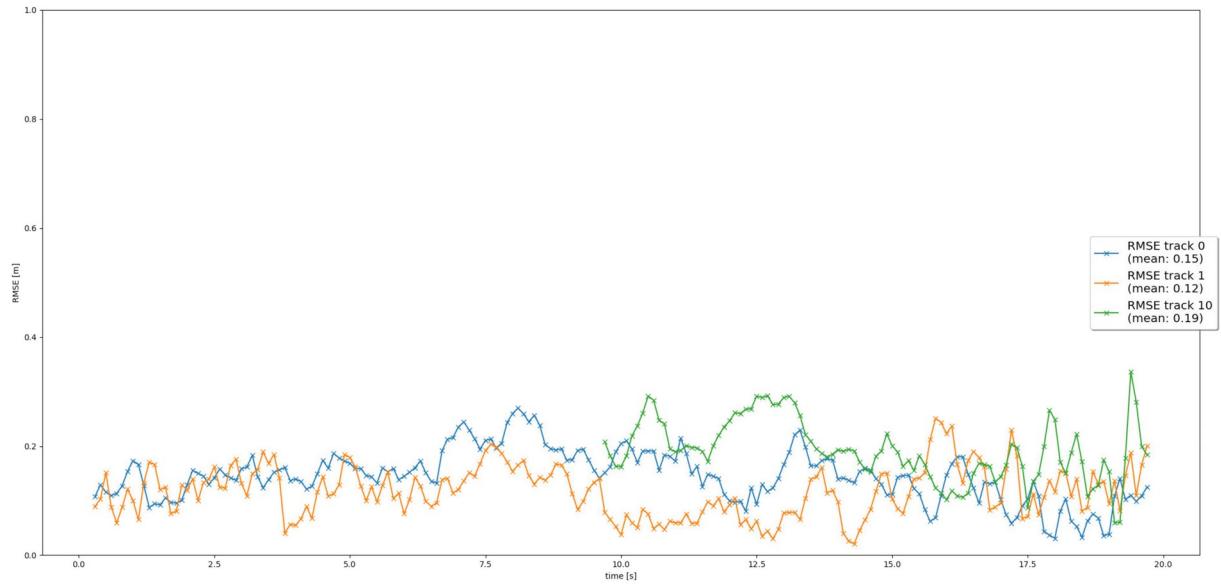
The changes are inside the "association.py" file:

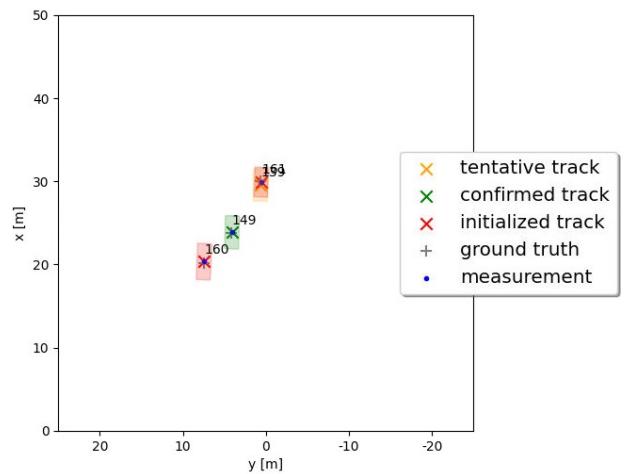
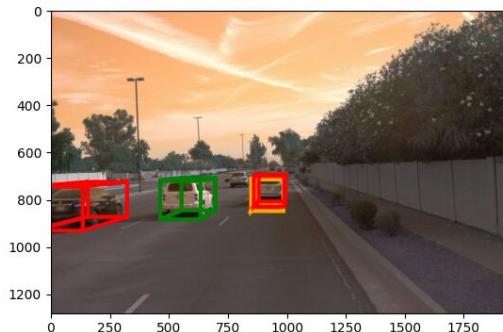
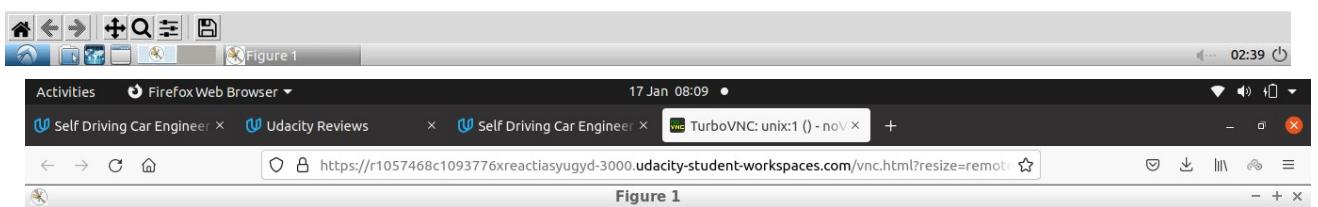
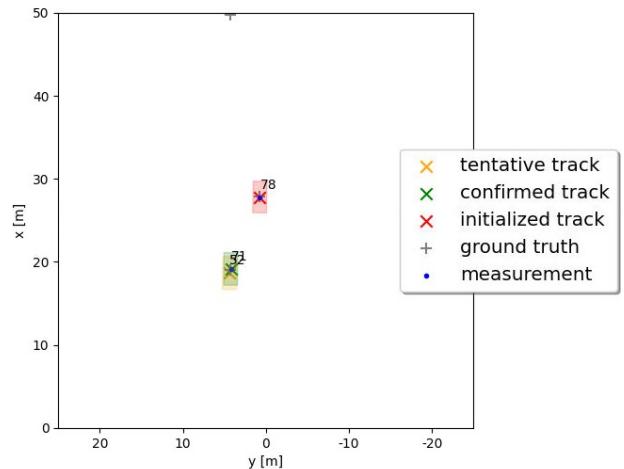
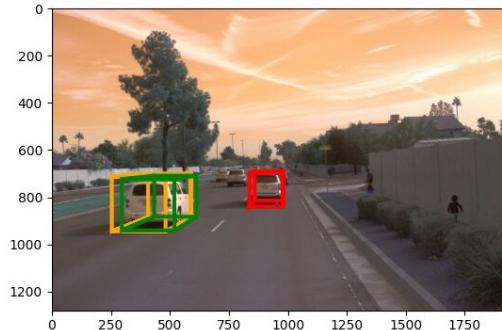
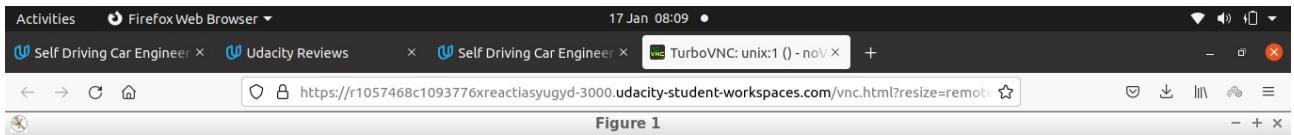


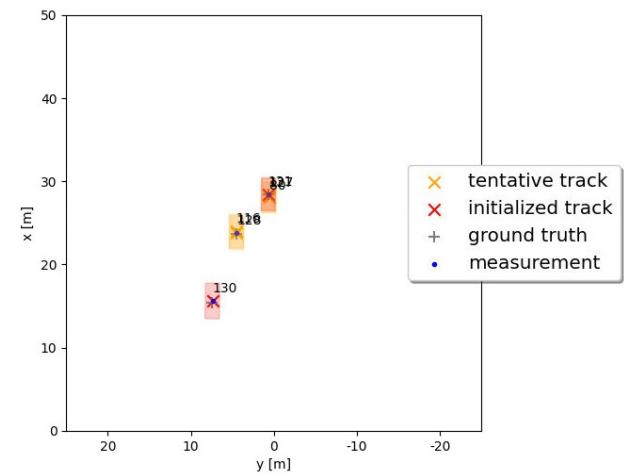
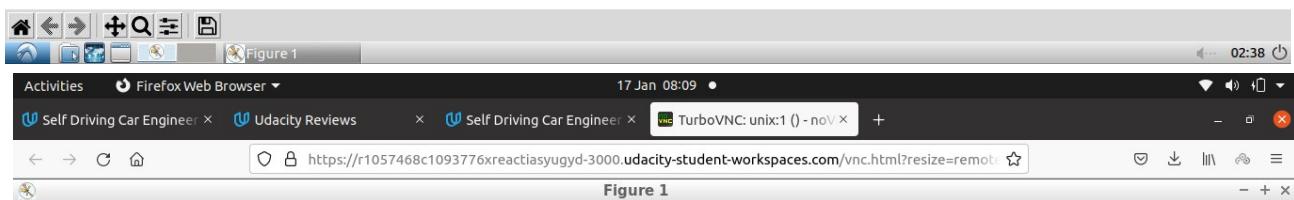
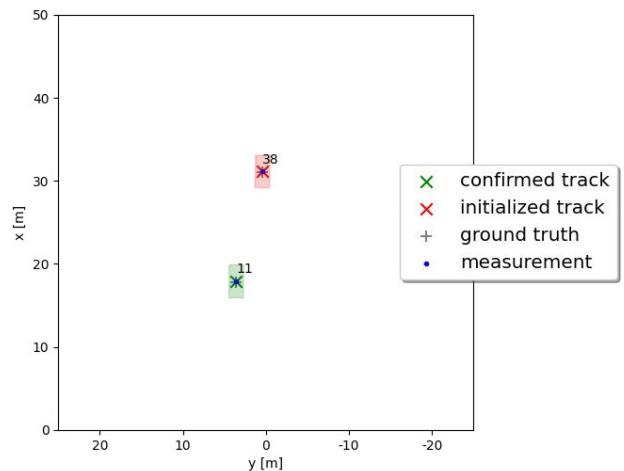
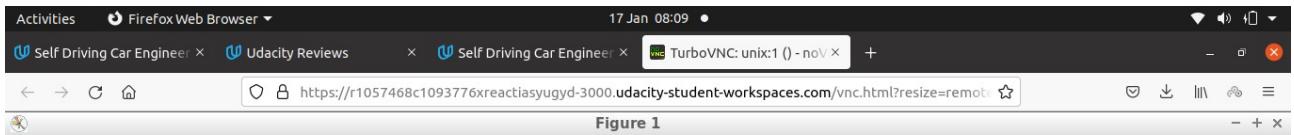
A screenshot of a Firefox browser window titled "Tracking Workspace". The address bar shows the URL <https://classroom.udacity.com/nanodegrees/nd0013/part/e09820a6-7410-4d72-aba2-a442f5>. The workspace contains several tabs: README.md, loop_over_dataset.py, association.py (which is currently active), measurements.py, filter.py, and trackmanagement.py. The association.py tab displays the following Python code:

```
32 def associate(self, track_list, meas_list, KF):
33     #####
34     # TODO Step 3: association:
35     # - replace association matrix with the actual association matrix based on Mahalanobis distance (see below) for all tracks and all measurements
36     # - update list of unassigned measurements and unassigned tracks
37     #####
38
39     num_tracks = len(track_list)
40     num_meas = len(meas_list)
41
42
43     # the following only works for at most one track and one measurement
44     self.association_matrix = np.inf*np.ones((num_tracks, num_meas)) # initialize association matrix
45     self.unassigned_tracks = list(range(num_tracks)) # reset lists
46     self.unassigned_meas = list(range(num_meas))
47
48
49     """if len(meas_list) > 0:
50         self.unassigned_meas = []
51     if len(track_list) > 0:
52         self.unassigned_tracks = []
53     if len(meas_list) > 0 and len(track_list) > 0:
54         self.association_matrix = np.matrix([[0]])
55
56         for t in range(num_tracks):
57             track = track_list[t]
58             for m in range(num_meas):
59                 meas = meas_list[m]
60                 dist = self.MHD(track, meas, KF)
61                 if self.gating(dist, meas.sensor):
62                     self.association_matrix[t][m] = dist
63
64
```

A sample preview of the association measurement:







```

Activities Firefox Web Browser • 16 Jan 22:38 •
Self Driving Car Engin X Self Driving Car Engin X TurboVNC: unix:1 () - X AttributeError: modu X Common error mess X [Solved] AttributeErr X + - _ X

Tracking Workspace
SEND FEEDBACK
/ > home > workspace > misc
__pycache__
evaluation.py
helpers.py
objdet_tools.py
params.py
122 f_frame = None
+ BASH
loading object labels and validation from result file
loading detection performance measures from file
predict track 174
predict track 201
predict track 201
predict track 202
lidar chisqr = 1.0
lidar chisqr = 1.0
lidar chisqr = 0.2854520834429309
lidar chisqr = 0.9988083252189357
lidar chisqr = 1.0
lidar chisqr = 1.0
lidar chisqr = 1.0
lidar chisqr = 0.998105105454898
lidar chisqr = 1.0
update track 174 with lidar measurement 2
deleting track no. 201
creating track no. 203
creating track no. 204
track 174 score = 2.666666666666666
track 202 score = 0.0
track 203 score = 0.1666666666666666
track 204 score = 0.1666666666666666
deleting track no. 202
track 174 score = 2.666666666666666
track 203 score = 0.1666666666666666
track 204 score = 0.1666666666666666
StopIteration has been raised
(sdc-c2) root@285072775df4:/home/workspace#

```

DESKTOP NEXT

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:

```

Tracking Workspace
README.md      X  loop_over_data...  X
76
77 camera = None # init camera sensor object
78 np.random.seed(10) # make random values predictable
79
80 ## Selective execution and visualization
81 exec_detection = [] #'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_performance'] # options are 'bev_from_
    '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', 'make_tracking_movie'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image',
    'show_objects_and_labels_in_bev', '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

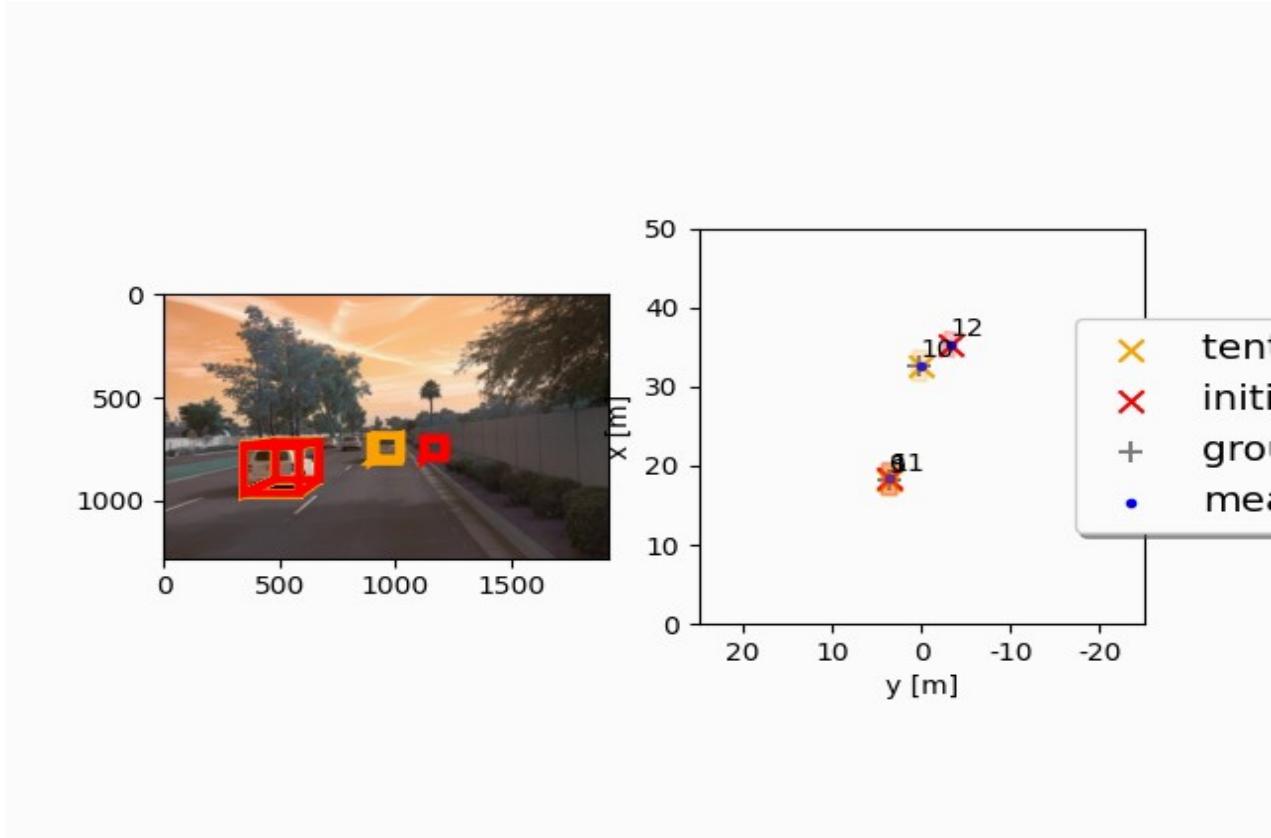
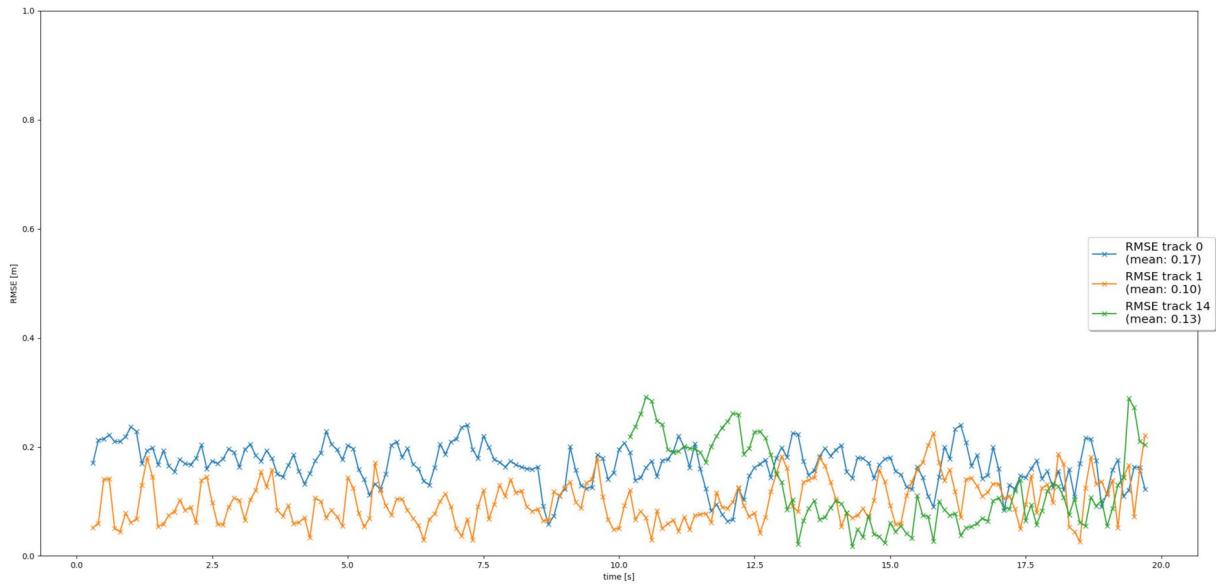
```

The changes for "measurements.py" :

```
Activities Firefox Web Browser ▾ 19 Jan 22:17
Self Driving Car × Udacity Reviews × Udacity Reviews × Self Driving Car × Self Driving Car × Knowledge - Uda× New Tab × + - ×
← → ⌂ ⌂ https://classroom.udacity.com/nanodegrees/nd0013/part... 90% ☆
☰ Tracking Workspace SEND FEEDBACK
README.md × loop_over_datal... × association.py × measurements.py × filter.py × trackmanagement... ×
43 </> home>workspace>student
44 def in_fov(self, x):
45     # check if an object x can be seen by this sensor
46     #####
47     # TODO Step 4: implement a function that returns True if x lies in the sensor's field of view,
48     # otherwise False.
49     #####
50     pos_veh = np.ones((4, 1)) # homogeneous coordinates
51     pos_veh[0:3] = x[0:3]
52     pos_sens = self.veh_to_sens*pos_veh # transform from vehicle to lidar coordinates
53     x,y,z = np.squeeze(pos_sens.A)[0:3]
54     if self.name == "Lidar":
55         # angle = math.atan2(y, x)
56     else:
57         # angle = math.atan2(y, x)
58     # print("camera fov={}".format(angle))
59     # to avoid division by zero
60     if pos_sens[0] > 0:
61         alpha = np.arctan(pos_sens[1]/pos_sens[0])
62         if alpha > self.fov[0] and alpha<self.fov[1]:
63             return True
64     return False
65
66     #####
67
68     #####
69     # END student code
70     #####
71
+ BASH ×
DESKTOP NEXT
↑ Menu
```

```
Activities Firefox Web Browser ▾ 19 Jan 22:18
Self Driving Car × Udacity Reviews × Udacity Reviews × Self Driving Car × Self Driving Car × Knowledge - Uda× New Tab × + - ×
← → ⌂ ⌂ https://classroom.udacity.com/nanodegrees/nd0013/part... 90% ☆
☰ Tracking Workspace SEND FEEDBACK
README.md × loop_over_datal... × association.py × measurements.py × filter.py × trackmanagement... ×
72 </> home>workspace>student
73 def get_hx(self, x):
74     # calculate nonlinear measurement expectation value h(x)
75     if self.name == 'lidar':
76         pos_veh = np.ones((4, 1)) # homogeneous coordinates
77         pos_veh[0:3] = x[0:3]
78         pos_sens = self.veh_to_sens*pos_veh # transform from vehicle to lidar coordinates
79         return pos_sens[0:3]
80     elif self.name == 'camera':
81         #####
82         # TODO Step 4: implement nonlinear camera measurement function h:
83         # - transform position estimate from vehicle to camera coordinates
84         # - project from camera to image coordinates
85         # - make sure to not divide by zero, raise an error if needed
86         # - return h(x)
87         #####
88         # transform from vehicle to lidar coordinates
89         pos_veh = np.zeros((2, 1)) # homogeneous coordinates
90
91         if x[0]==0:
92             raise NameError('Jacobian not defined for x[0]=0!')
93         else:
94             pos_veh[0,0] = self.c_i - self.f_i*x[1]/x[0] # project to image coordinates
95             pos_veh[1,0] = self.c_j - self.f_j*x[2]/x[0]
96             return pos_veh
97
98         #pass
99
100
+ BASH ×
DESKTOP NEXT
↑ Menu
```

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.