YILDIZ TECHNICAL UNIVERSITY

MKT 4846 – INTRODUCTION TO AUTONOMOUS DRIVING CAR

KEMALETTIN KARA

1806A004 MAIN PROJECT

LECTURER: AYDIN YESILDIREK

MKT4846 Sensor Fusion and Tracking Project

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.

## Project File Structure

📦project

┣ 📂dataset --> contains the Waymo Open Dataset sequences <br>

┃

┣ 📂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 from lidar and darknet

```
│
├ 📂deliverables
│ ├ 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
│ │ │
│ │ ├ 📂darknet
│ │ │ ├ 📂config
│ │ │ ├ 📂models --> darknet / yolo model class and tools
│ │ │ ├ 📂pretrained --> copy pre-trained model file here
│ │ │ │ └ complex_yolov4_mse_loss.pth
│ │ │ ├ 📂utils --> various helper functions
│ │ │
│ │ └ 📂resnet
```

┃ ┃ ┃ ┣ 📁models --> fpn_resnet model class and tools

┃ ┃ ┃ ┣ 📁pretrained --> copy pre-trained model file here

┃ ┃ ┃ ┃ ┗ fpn_resnet_18_epoch_300.pth

┃ ┃ ┃ ┣ 📁utils --> various helper functions

┃ ┗ ┃

┃

┣ loop_over_dataset.py

Running of the project

In classroom “submit the zipped source codes that you have edited” enounced. I assume you have tools and other unedited files. In here just extract zipped project file and run loop_over_dataset.py If you take torch.utils error I solve with remove utils.py from python built-in files. Protobuf error shown in below.

## Installation Instructions

### Python

The project has been written using Python 3.10.9. Please make sure that your local installation is above Python 3.7 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`.

## WARNING: Some code could be from python2 or old version and protobuf gives error in latest version in my pc if you take any error when you install waymo reader please install protobuf 3.20.* versions.

### Waymo Open Dataset Files

This project makes use of two 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.

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 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`.

### WARNING: When unzip the files please put them all just training files not folder because of some code takes the path and cannot find them be careful with file path gives an error.

- `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. [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)

***Warning: Please put all darknet results and Lidar results in same file as extracted (not in darknet folder or lidar folder)

### ERRORS WHICH I TOOK

1 -) When required torch.utils does not work

# Control version of torch.__version__ I use CUDA 11.6

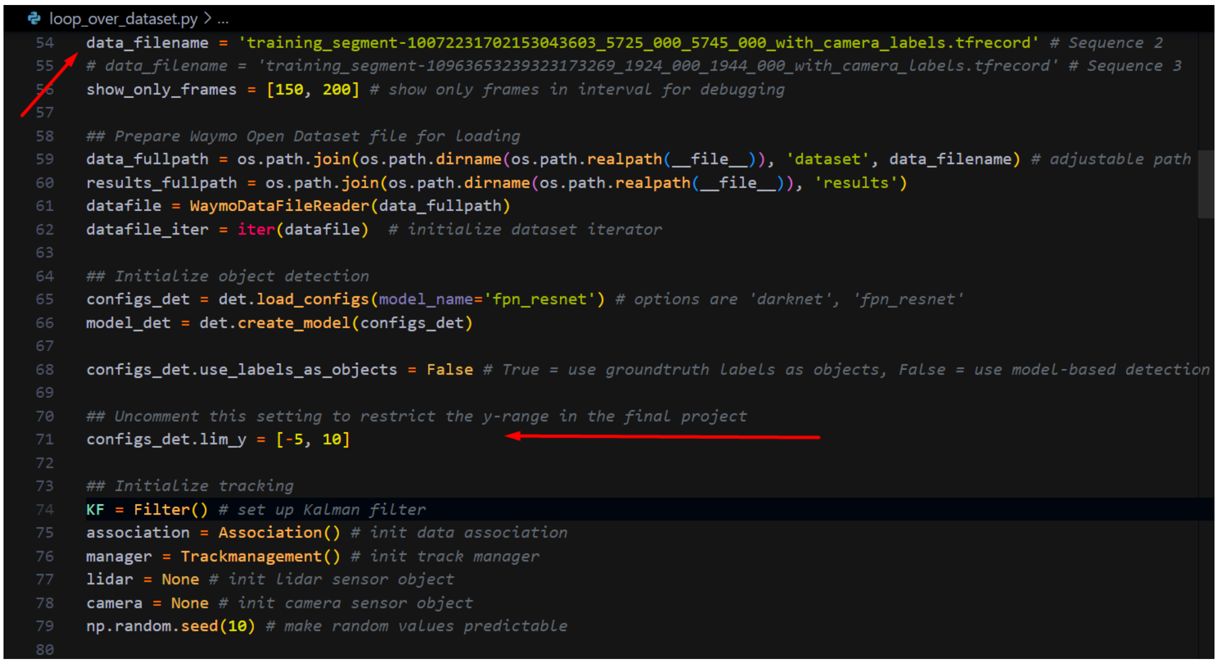
# Could be duplication and python env. goes utils.py. Python confuse utils use remove utils.py from CMD and run the again.

# If easydict gives error for pretrained model Resnet or Darknet whatever you can use python virtual env. or other equivalent released library like

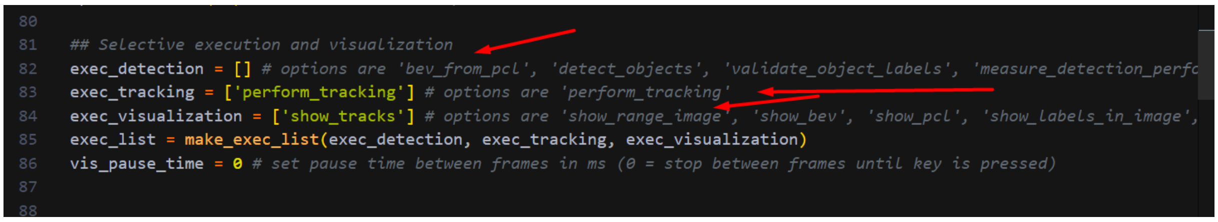
MY RESULTS

STEP1: Implement an extended Kalman filter.

- Select Sequence 2 (training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord) by uncommenting this line in loop_over_dataset.py and commenting the other sequences.
- Set show_only_frames = [150, 200] in order to limit the sequence to frames 150 to 200. This is the time span where our single object is visible.
- Set configs_det = det.load_configs(model_name='fpn_resnet') to use the Resnet neural network architecture. Note that Darknet is not applicable here because it does not estimate the height.
- Set configs_det.lim_y = [-5, 10] to limit the y-range and remove other targets left and right of our target. To do so, uncomment the respective line.
- Set exec_detection = [] to skip the lidar detection for faster execution and to load the lidar results from file instead.
- Set exec_tracking = ['perform_tracking'] to activate tracking.
- Set exec_visualization = ['show_tracks'] for track visualization.


loop_over_dataset.py > ...

```python
data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
# data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [150, 200] # show only frames in interval for debugging

## Prepare Waymo Open Dataset file for loading
data_fullpath = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'dataset', data_filename) # adjustable path
results_fullpath = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'results')
datafile = WaymoDataFileReader(data_fullpath)
datafile_iter = iter(datafile)  # initialize dataset iterator

## Initialize object detection
configs_det = det.load_configs(model_name='fpn_resnet') # options are 'darknet', 'fpn_resnet'
model_det = det.create_model(configs_det)

configs_det.use_labels_as_objects = False # True = use groundtruth labels as objects, False = use model-based detection

## Uncomment this setting to restrict the y-range in the final project
configs_det.lim_y = [-5, 10]

## Initialize tracking
KF = Filter() # set up Kalman filter
association = Association() # init data association
manager = Trackmanagement() # init track manager
lidar = None # init lidar sensor object
camera = None # init camera sensor object
np.random.seed(10) # make random values predictable

```



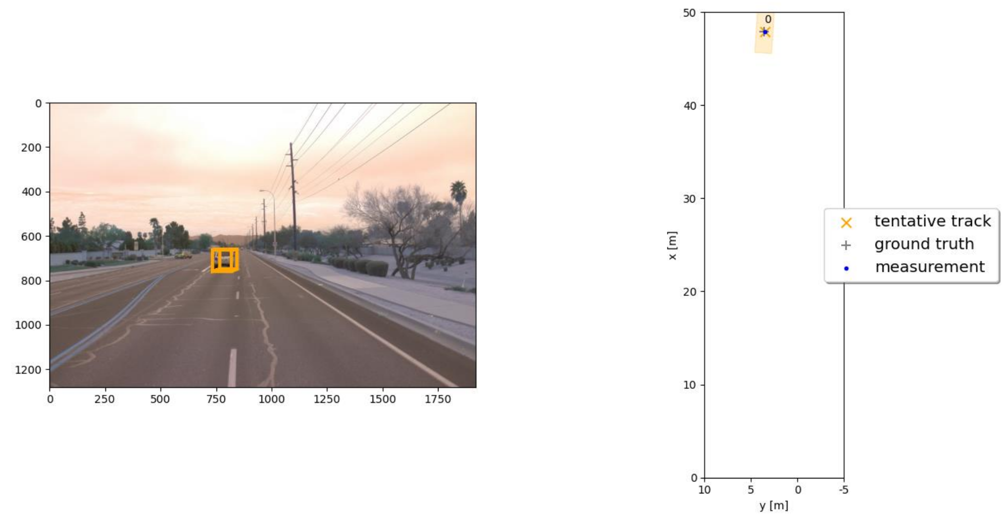
```python

## Selective execution and visualization
exec_detection = [] # options are 'bev_from_pcl', 'detect_objects', 'validate_object_labels', 'measure_detection_perfo
exec_tracking = ['perform_tracking'] # options are 'perform_tracking'
exec_visualization = ['show_tracks'] # options are 'show_range_image', 'show_bev', 'show_pcl', 'show_labels_in_image',
exec_list = make_exec_list(exec_detection, exec_tracking, exec_visualization)
vis_pause_time = 0 # set pause time between frames in ms (0 = stop between frames until key is pressed)


```

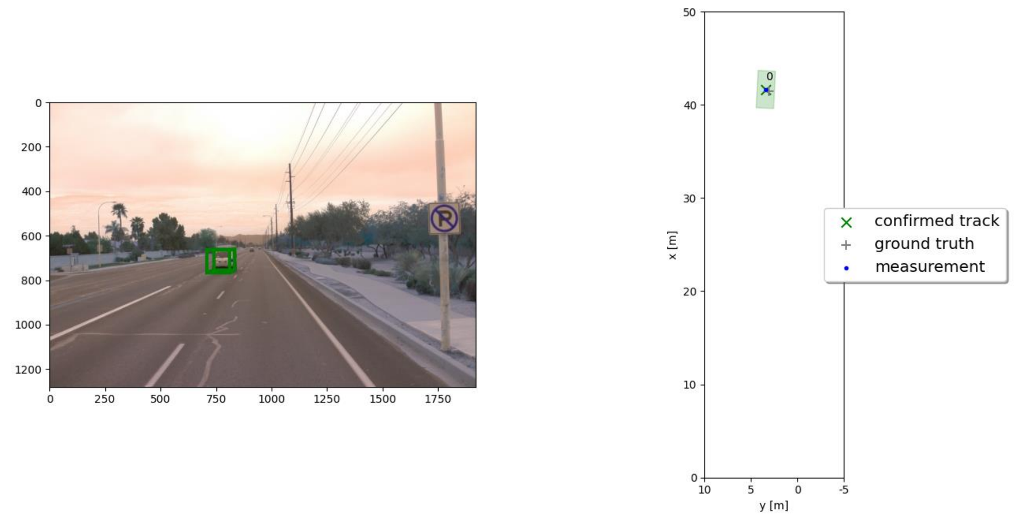
- The single track is already initialized for you, so don't worry about track initialization right now.
- In student/filter.py, implement the predict() function for an EKF. Implement the F() and Q() functions to calculate a system matrix for constant velocity process model in 3D and the corresponding process noise covariance depending on the current timestep dt. Note that in our case, dt is fixed and you should load it from misc/params.py. However, in general, the timestep might vary. At the end of the prediction step, save the resulting x and P by calling the functions set_x() and set_P() that are already implemented in student/trackmanagement.py.

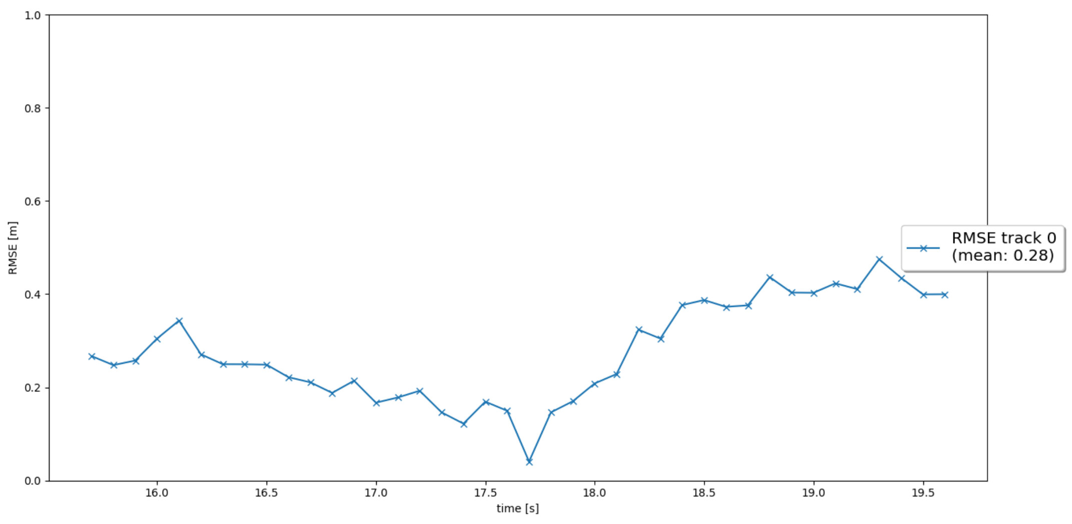
  - Implement the update() function as well as the gamma() and S() functions for residual and residual covariance. You should call the functions get_hx and get_H that are already implemented in students/measurements.py to get the measurement function evaluated at the current state, h(x), and the Jacobian H. Note that we have a linear measurement model for lidar, so h(x)=H*x for now. You should use h(x) nevertheless for the residual to have an EKF ready for the nonlinear camera measurement model you'll need in Step 4. Again, at the end of the update step, save the resulting x and P by calling the functions set_x() and set_P() that are already implemented in student/trackmanagement.py.
  - Use numpy.matrix() for all matrices as learned in

  - If you have implemented everything correctly, the RMSE plot should show a mean RMSE of 0.35 or smaller. You can see the computed mean RMSE in the legend on the right. Make sure to successfully complete this step and save the RMSE plot before moving to the next.

In 197 frame iteration my output is below.







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

We now want to track 3D objects with a constant velocity model including height estimation, so F and Q will be 6D matrices, in comparison to our 2D tracking in the lesson exercise where we assumed a flat world. Therefore, you need to implement the following matrices: [TODO: include image of Latex formulas]

● Remember from the repository overview on the last page that there is a Track class and a Measurement class. These classes define your input to the predict() and update() functions. Soyou can get the track data by calling track.x and track.P, the measurement data by calling meas.z and meas.R. Also note that the measurement has an attribute sensor that tells us which sensor generated this measurement, so you can get the measurement matrix by calling meas.sensor.get_H(). Take a closer look at the two classes for clarification.

● Note that you don't have a running track management yet, therefore the track state is fixed at 'confirmed' and the score remains at the initial value zero.

● From misc/params.py, you should load the following parameters: dt, q, dim_state

Project Instructions Step 2

● Make sure to refer to the project rubric to ensure all tasks are completed.

● What is this task about?

● In Step 2 of the final project, you will implement the track management to initialize and delete tracks, set a track state and a track score.

● Task preparation

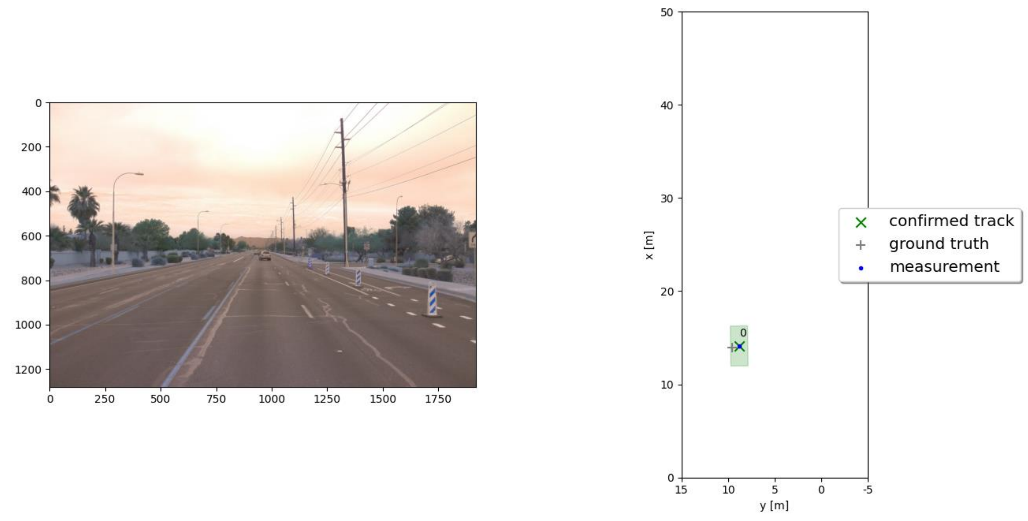
● In addition to the settings from Step 1, apply the following settings in loop_over_dataset.py:

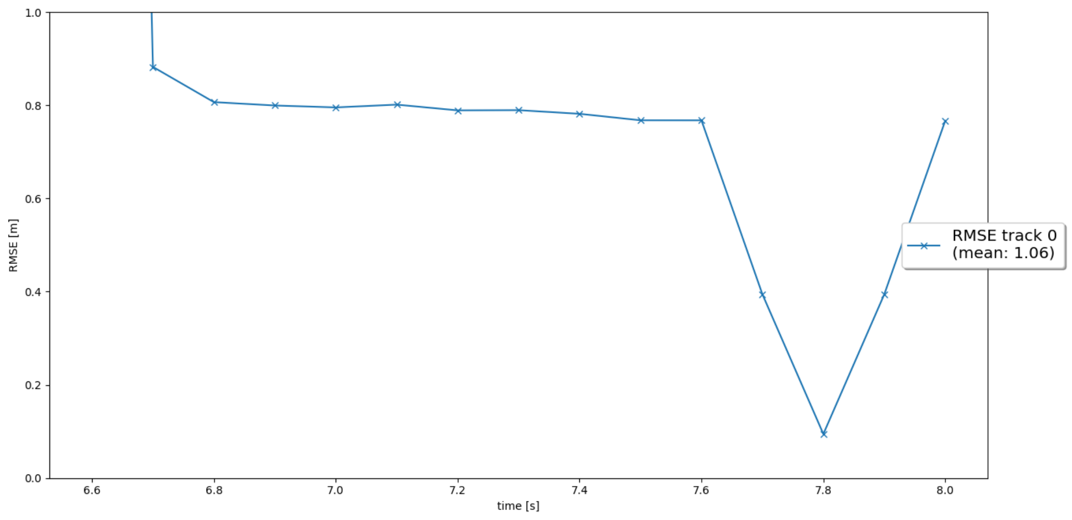
Set show_only_frames = [65, 100] in order to limit the sequence to frames 65 to 100. This is the time span where a single object appears and then disappears, so we can use it for track initialization and deletion.

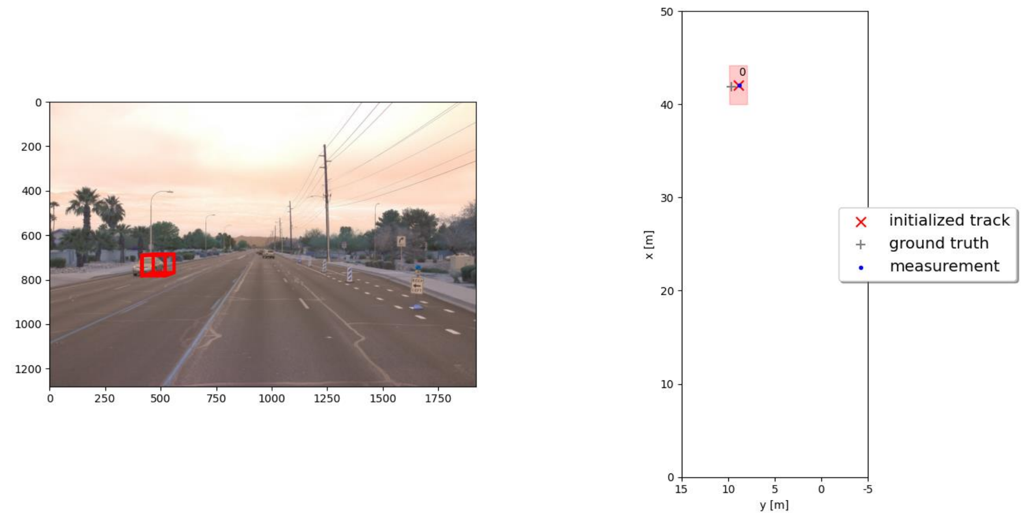
● Set configs_det.lim_y = [-5, 15] to limit the y-range and remove other targets left and right of our target.

● Where to find this task?

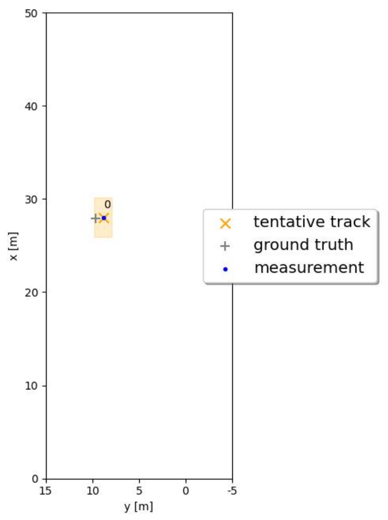
● This task involves writing code within the file student/trackmanagement.py. Please ignore TODOs for other steps in other files for now.



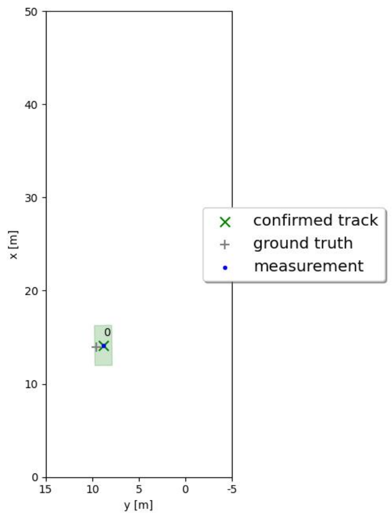




0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


50
40
30
20
10
0
x [m]
15
10
5
0
-5
y [m]
0
tentative track
ground truth
measurement

0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


50
40
30
20
10
0
x [m]
15
10
5
0
-5
y [m]
0
confirmed track
ground truth
measurement

In Step 3 of the final project, you will implement a single nearest neighbor data association to associate measurements to tracks. You will finally move on to multi target tracking now!

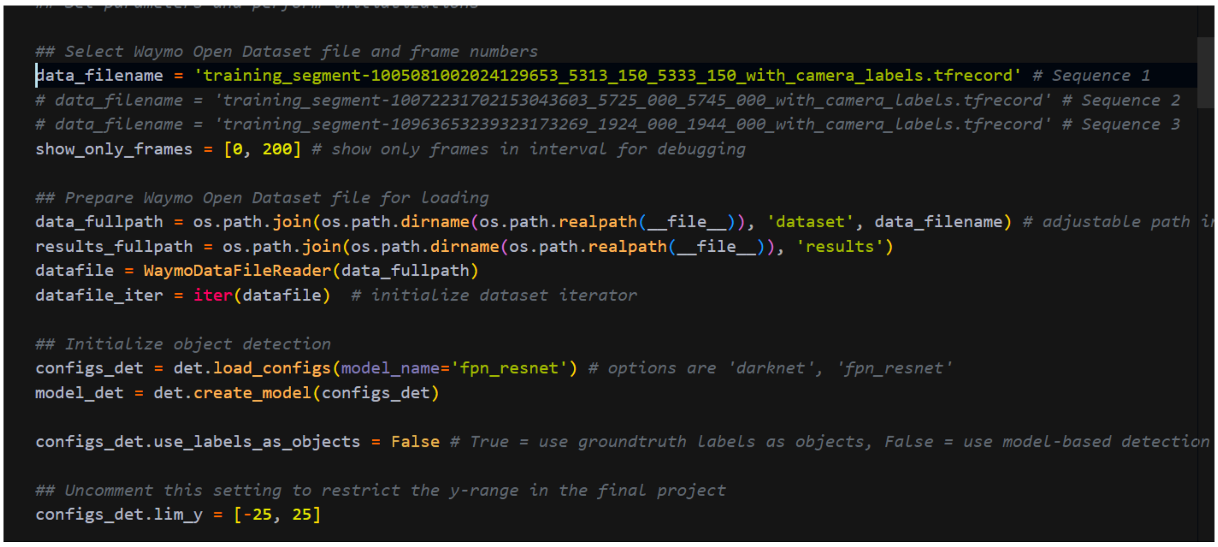
● Task preparation

● In addition to the settings from Step 2, apply the following settings in loop_over_dataset.py:

● Select Sequence 1 (training_segment-10050810020241296535_5313_150_5333_150_with_camera_labels.tfrecord) by uncommenting this line in loop_over_dataset.py and commenting out the other sequences. This is a more complex scenario with multiple targets.

● Set show_only_frames = [0, 200] in order to use the whole sequence now.

● Set configs_det.lim_y = [-25, 25] to use the whole y-range including several targets.



```
## Select Waymo Open Dataset file and frame numbers
data_filename = 'training_segment-1005081002024129653_5313_150_5333_150_with_camera_labels.tfrecord' # Sequence 1
# data_filename = 'training_segment-10072231702153043603_5725_000_5745_000_with_camera_labels.tfrecord' # Sequence 2
# data_filename = 'training_segment-10963653239323173269_1924_000_1944_000_with_camera_labels.tfrecord' # Sequence 3
show_only_frames = [0, 200] # show only frames in interval for debugging

## Prepare Waymo Open Dataset file for loading
data_fullpath = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'dataset', data_filename) # adjustable path i
results_fullpath = os.path.join(os.path.dirname(os.path.realpath(__file__)), 'results')
datafile = WaymoDataFileReader(data_fullpath)
datafile_iter = iter(datafile)  # initialize dataset iterator

## Initialize object detection
configs_det = det.load_configs(model_name='fpn_resnet') # options are 'darknet', 'fpn_resnet'
model_det = det.create_model(configs_det)

configs_det.use_labels_as_objects = False # True = use groundtruth labels as objects, False = use model-based detection

## Uncomment this setting to restrict the y-range in the final project
configs_det.lim_y = [-25, 25]
```

Your task

● In the Association class, implement the associate() function to complete the following tasks:

● Replace association_matrix with the actual association matrix based on Mahalanobis distances for all tracks in the input track_list and all measurements in the input meas_list. Use the MHD() function to implement the Mahalanobis distance between a track and a measurement. Also, use the gating() function to check if a measurement lies inside a track's gate. If not, the function shall return False and the entry in association_matrix shall be set to infinity.

● Update the list of unassigned measurements unassigned_meas and unassigned tracks unassigned_tracks to include the indices of all measurements and tracks that did not get associated.

In the Association class, implement the get_closest_track_and_meas() function to complete the following tasks:

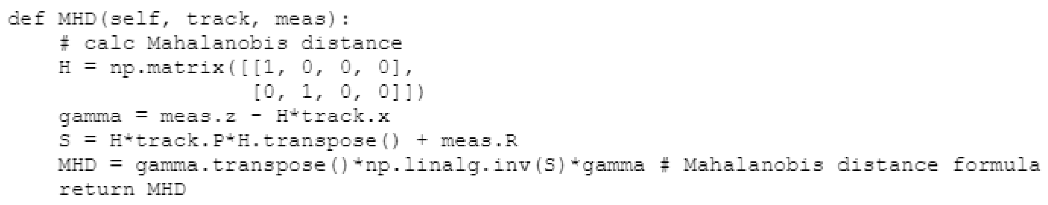
● Find the minimum entry in association_matrix, delete corresponding row and column from the matrix.

● Remove corresponding track and measurement from unassigned_tracks and unassigned_meas.

● Return this association pair between track and measurement. If no more association was found, i.e. the minimum matrix entry is infinity, return numpy.nan for the track and measurement.

NOTE:

In my building some training camera labels cannot read. I checked several times but there is no error. I think because of running of software is slow. I found some stackoverflow but there is no way to fix it which I can do.



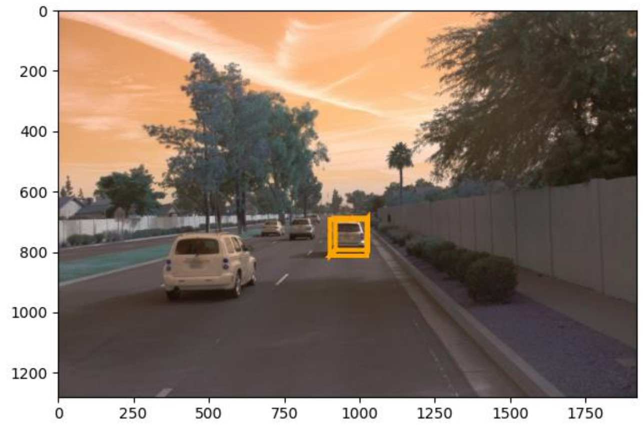
```
def MHD(self, track, meas):
    # calc Mahalanobis distance
    H = np.matrix([[1, 0, 0, 0],
                   [0, 1, 0, 0]])
    gamma = meas.z - H*track.x
    S = H*track.P*H.transpose() + meas.R
    MHD = gamma.transpose()*np.linalg.inv(S)*gamma # Mahalanobis distance formula
    return MHD
```

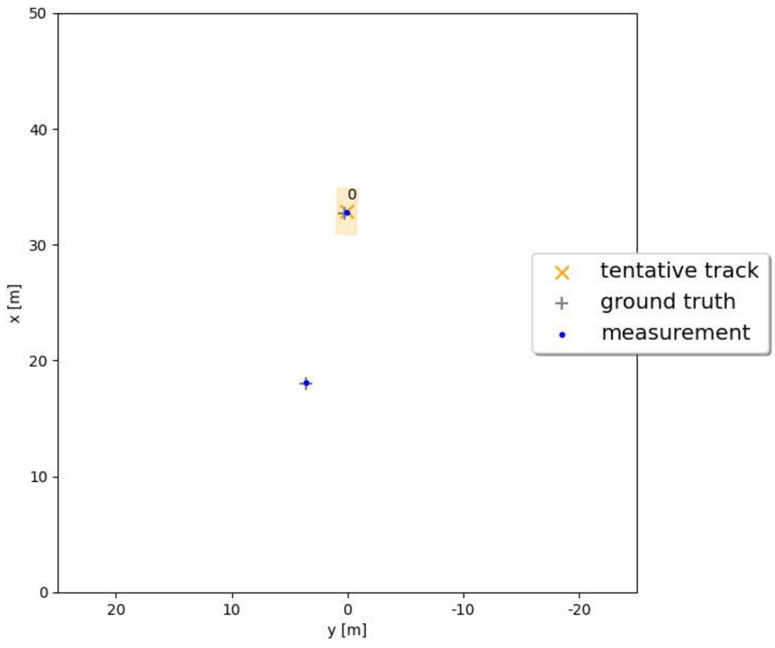
I do some code from code illustration which shared in classroom (tracking illustration) but I can’t fix. Python suggest run with Python discriminator but program will so slow, then I continue.

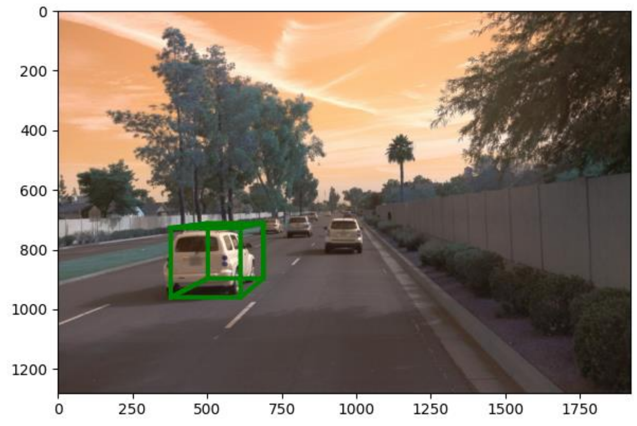
The association works properly if you see in the visualization that multiple tracks are updated with multiple measurements. The console output shows that each measurement is used at most once and each track is updated at most once. The visualization should show that there are no confirmed “ghost tracks” that do not exist in reality. There may be initialized or tentative “ghost tracks” as long as they are deleted after several frames. Make sure to successfully complete this step and save the RMSE plot before moving to the next. If you still saw some initialized or tentative ghost tracks here, let's see if we can deplausibilize them through sensor fusion with camera in the next step!

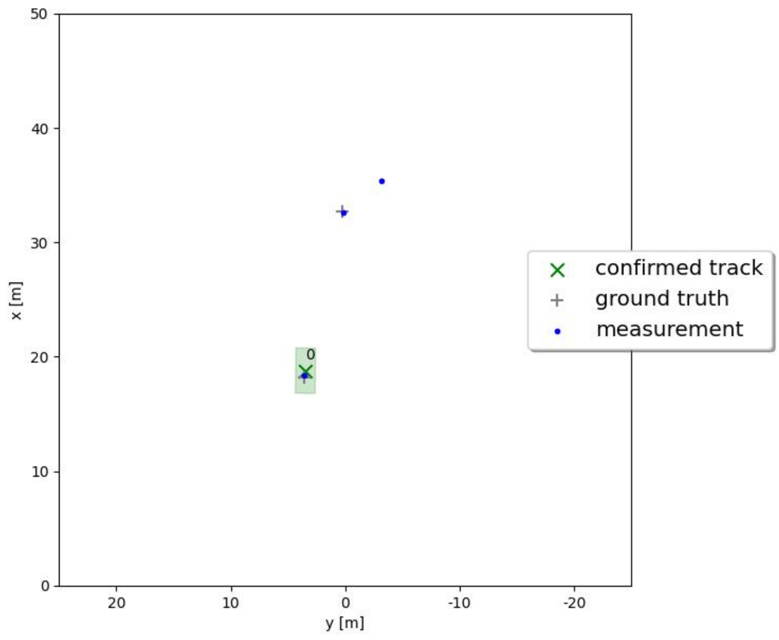
MY RESULT

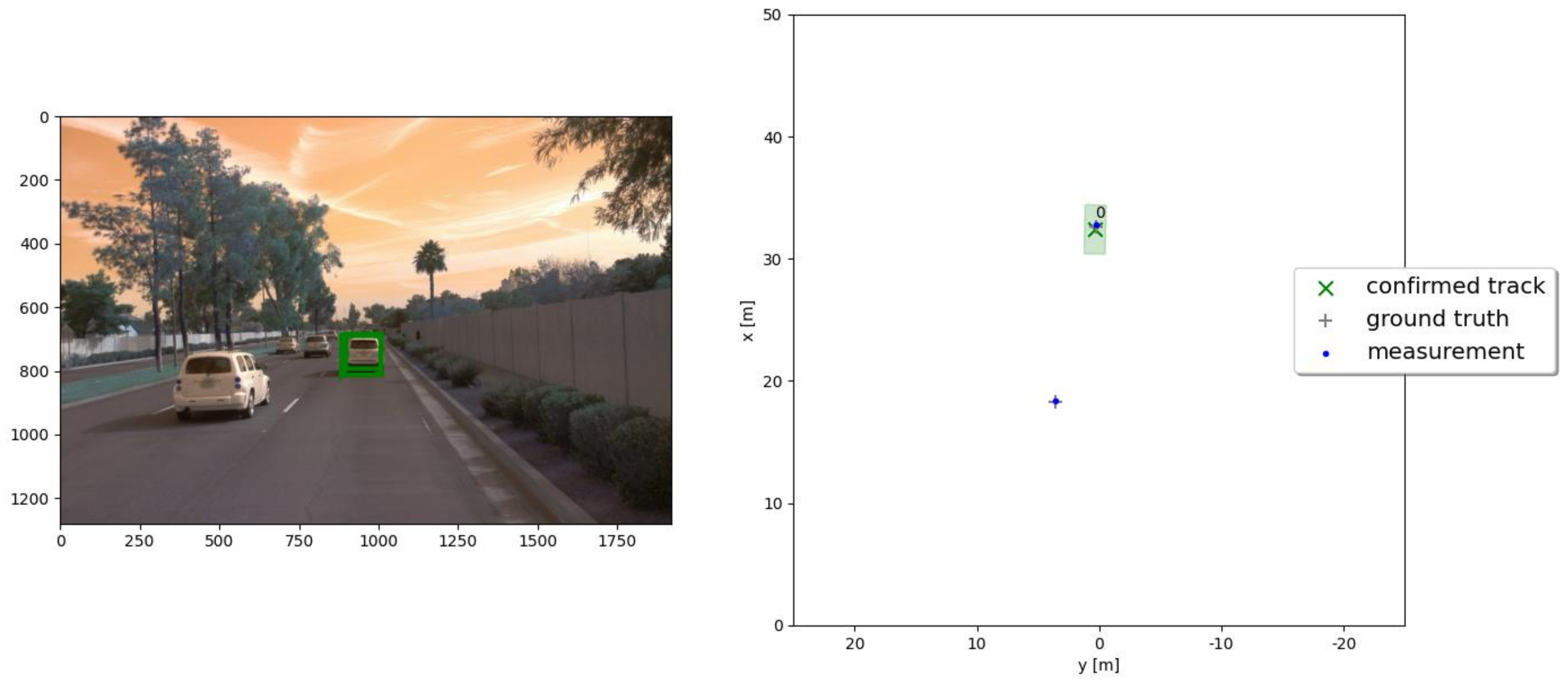
Console output doesn't show that a single metric has been used multiple times so I've deduced that row and column deletions are working correctly. No confirmed tracks found to plot RMSE! Sir I try many times in step 3 I take RMS but I couldn't track multiple vehicles.



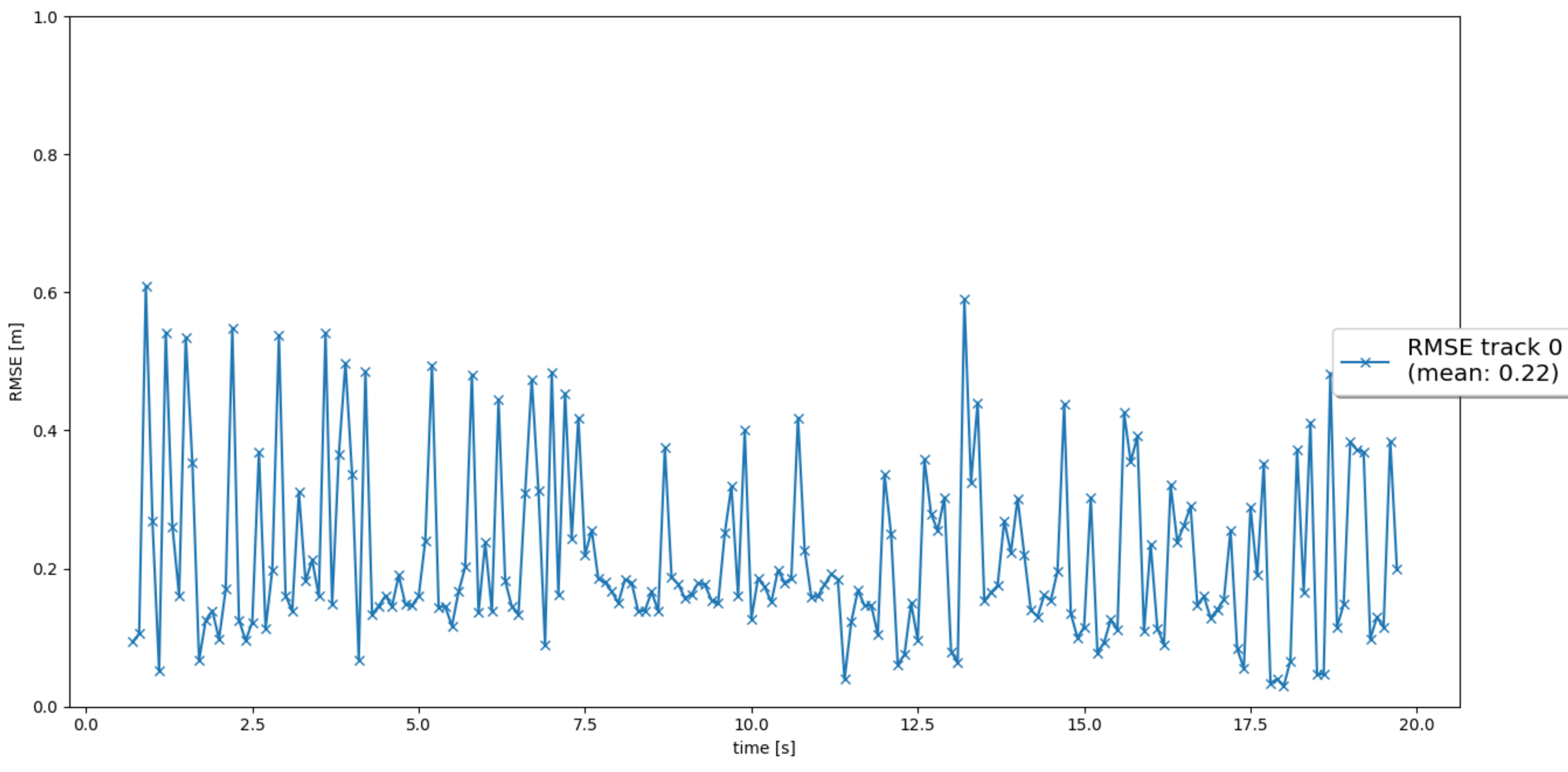








0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750
50
40
30
20
10
0
x [m]
20
10
0
-10
-20
y [m]
0
confirmed track
ground truth
measurement



1.0
0.8
0.6
0.4
0.2
0.0
RMSE [m]
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
time [s]
RMSE track 0
(mean: 0.22)

Project Instructions Step 4

● Make sure to refer to the project rubric to ensure all tasks are completed.

● What is this task about?

● In Step 4 of the final project, you will implement the nonlinear camera measurement model. You will finally complete the sensor fusion module for camera-lidar fusion!

● Task preparation

● The settings are the same as for Step 3.

● Where to find this task?

● This task involves writing code within the file student/measurements.py.

Your task

● In the Sensor class, implement the function in_fov() that checks if the input state vector x of an object can be seen by this sensor. The function should return True if x lies in the sensor's field of view, otherwise False. Don't forget to transform from vehicle to sensor coordinates first. The sensor's field of view is given in the attribute fov.

● In the Sensor class, implement the function get_hx() with the nonlinear camera measurement function h as follows:

● transform position estimate from vehicle to camera coordinates,

● project from camera to image coordinates,

● make sure to not divide by zero, raise an error if needed,

● return h(x).

In the Sensor class, simply remove the restriction to lidar in the function generate_measurement() in order to include camera as well.

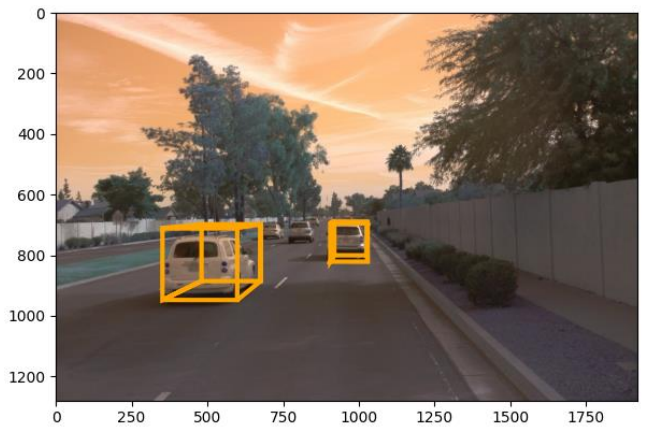
● In the Measurement class, initialize camera measurement objects including z, R, and the sensor object sensor.

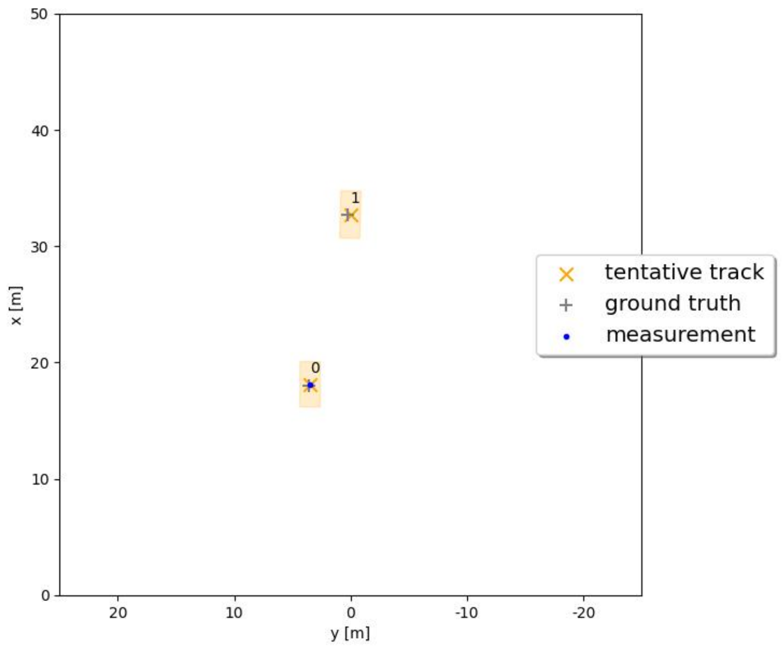
● After completing these steps, make a movie to showcase your tracking results! You can simply do so by setting exec_visualization = ['show_tracks', 'make_tracking_movie'] in loop_over_dataset.py and re-running the tracking loop.

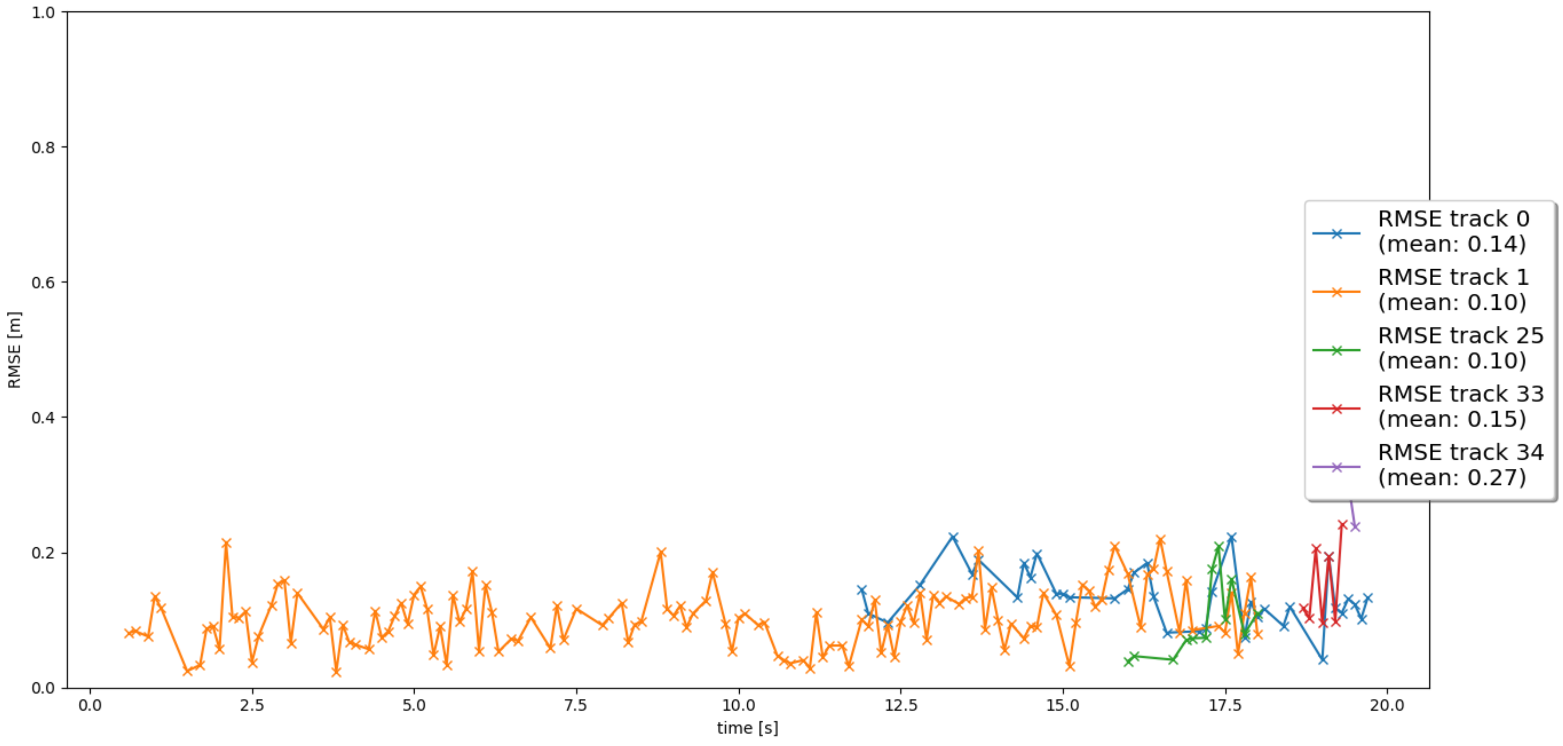
● What should the result be?

● If you have implemented everything correctly, the tracking loop now updates all tracks with lidar measurements, then with camera measurements. The console output shows lidar updates followed by camera updates. The visualization shows that the tracking performs well, again no confirmed ghost tracks or track losses should occur. The RMSE plot should show at least three confirmed tracks. Two of the tracks should be tracked from beginning to end of the sequence (0s -200s) without track loss. The mean RMSE for these two tracks should be below 0.25.

<u>MY RESULT</u>







There is a video in my video. I upload it to youtube please clik this link to watch video. Video output link.

Writeup Instructions

● 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?

During the homework, I tried to do it based on the codes we saw in the lesson, sometimes I got mistakes, especially in the Data association part, I had a lot of trouble. The reason for this was I could not find a source. There were many sources for EKF or tracking on the internet, but I could not understand or find much for association or sensor fusion, so I could not be fully successful.

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

-Yes, In terms of field of view, in terms of cost, and in terms of display, we have more attributes to make sense of the data. Text colors etc. I think that it is more difficult for the developer to interpret the outputs in the lidar assignment, which will create more time and labor while developing.

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

-The data association part was challenging. I don't think I could complete my homework. I constantly got errors in this part, sometimes there were shifts, I commented that it was caused by the dynamic environment, I don't know how true it is, I read many videos and articles in this area along with the homework. I've found sensor fusion to be difficult to adapt to real life problems, both because of the cost and I think it's because the environments are not the same everywhere. Especially when we compare places like Turkey and America, the environments are very different, the roads of the vehicles are very different. While the target of Eatron technology company in Turkey is level 2.5, I see that in autonomous vehicles such as Silicon Valley pittsburg, where startups are high, the targets are usually level 4, and they are sinking one by one because they cannot find investors and meet their targets.

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

-My thoughts during the assignment, although I'm not sure. I think that monitoring results can be improved by improving the optimization, hardware development and data association parts.