CS 498: Assignment 5: 3D Multi Object Tracking

Hongqing Liu, hl85

April 18, 2023

Submission

In this assignment, you will code a 3D multi-object tracking system based on a Kalman filter. You will be submitting two files, "kalman_filter.py" and "matching.py". Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course Code: BBX6NE). We recommend you add your answers to the latex template files we provided. More details on what to report are in the provided code.

Provided Files

Files you will modify:

- kalman_filter.py:
  - define_model (define dynamics)
  - update (observation model)
  - predict (state propagation)
- matching.py (greedy matching algorithm)
- main.py (for visualization and debugging)

Files you will not (need to) modify:

- evaluate.py: run this after running main.py to do evaluation, i.e. "python evaluate.py"
- kitti_calib/oxts.py: these are used for something called ego-motion-compensation, explained in code
- matching_utils.py: contains the function iou(box_a, box_b) which you will need to compute similarity when doing matching
- utils.py: some utils used by main.py, you should look at Box3D
- vis.py: some code for visualizing the data. You only really need vis_obj, which is described more clearly in main.py

File structure

```
data
├── calib
│   └── training
│       └── [seq_name].txt
├── detection
│   └── [seq_name].txt
├── label
│   └── [seq_name].txt
├── oxts
│   └── training
│       └── [seq_name].txt
└── image_02
    └── training
        └── [seq_name]
            └── [frame_num].png
results
├── eval (files in here will be created automatically)
└── img_vis (files in here will be created automatically)
```

Multi Object Tracking (MOT)

In this assignment, you will implement a multi object tracker for 3D objects. In other words, given a sequence of frames taken from the perspective of a car, track the other cars in the images. In this project we will develop our tracking algorithm on KITTI dataset(`http://www.cvlibs.net/datasets/kitti/`).

The idea is as follows: we can use a good pre-trained object detector to find objects in each frame (we've already done that for you, check `https://github.com/sshaoshuai/PointRCNN` if you want to know more). Now, simply identifying objects is not good enough, so we want to track them across frames for consistency. You will implement two modules which together will do this quite well.

The first is a matching algorithm: given a list of 3D bounding boxes for objects detected by the object detector in the current frame, match them to objects you are already tracking from the previous frame.

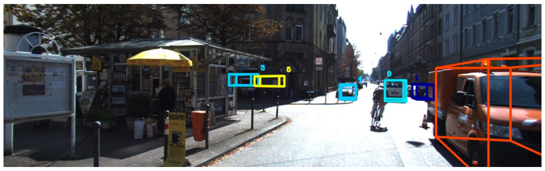
The second is a Kalman Filter. Just matching current detections to past trackers is not great since cars can move and therefore the previous and current bounding boxes will not overlap perfectly (this is especially problematic if an object becomes occluded for a few frames). To deal with this issue you will implement a Kalman Filter for forward propagating each object. Moreover, you will now use each object detection to "update" the state of your tracked object, as an observation to the filter.

Question 0 (Inspect the Data)[1 pt]: Before you begin the actual assignment, read the code we've provided. Namely read through "main.py" so you understand the basic structure of the algorithm and the functionality of each component. You will need to modify "main.py" for debugging/visualization purposes.

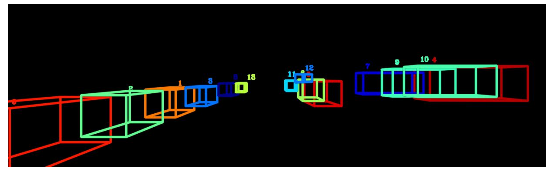
For this question, please visualize the detections for frame 100 of the first and second sequences, i.e. sequence 0000 and sequence 0001. Each detection should be a different color. You should work off the code we provided in the Visualization section of main.py.

Please download the images for the first sequence (0000), and put them in the right place in the directory structure above. Use your illinois address to access the drive link: `https://drive.google.com/file/d/15lWATvV4p9UCShnnPa2SEL8BTh2twIm4/view?usp=sharing`. This will allow you to overlay the sequence 0000 detections on top of the camera images for visualization.

Answers: The result of visualization are shown as follows:



(a) frame 100 in sequence 0000

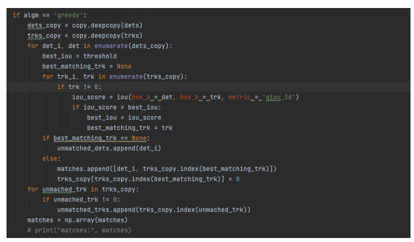


(b) frame 100 in sequence 0001

Question 1 (Greedy Matching)[3 pt]: For this question you will be modifying "matching.py". You should implement a greedy matching approach, whereby you match pairs of detections and tracked objects in order of similarity. First match the detection and tracker that are most similar, then remove them from the set, and continue, until you have no trackers or no detections. Also, if the similarity for a match is less than the provided threshold (-0.2), do not consider the pair a match.

Notice we provide you with "iou(box_a, box_b)" to compute similarity (in terms of generalized IoU) between 3D detected regions.

Answers: The codes are shown as follows:

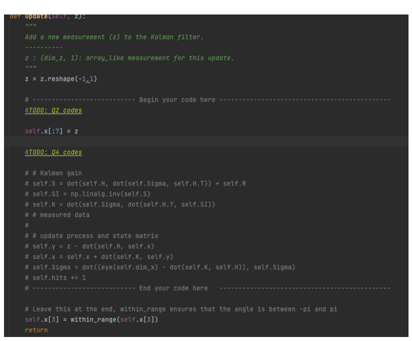


```
if algm == 'greedy':
    dets_copy = copy.deepcopy(dets)
    trks_copy = copy.deepcopy(trks)
    for det_i, det in enumerate(dets_copy):
        best_iou = threshold
        best_matching_trk = None
        for trk_i, trk in enumerate(trks_copy):
            if trk != 0:
                iou_score = iou(box_a = det, box_b = trk, metric = 'giou_3d')
                if iou_score > best_iou:
                    best_iou = iou_score
                    best_matching_trk = trk
        if best_matching_trk == None:
            unmatched_dets.append(det_i)
        else:
            matches.append([det_i, trks_copy.index(best_matching_trk)])
            trks_copy[trks_copy.index(best_matching_trk)] = 0
    for unmached_trk in trks_copy:
        if unmached_trk != 0:
            unmatched_trks.append(trks_copy.index(unmached_trk))
    matches = np.array(matches)
    # print("matches:", matches)
```

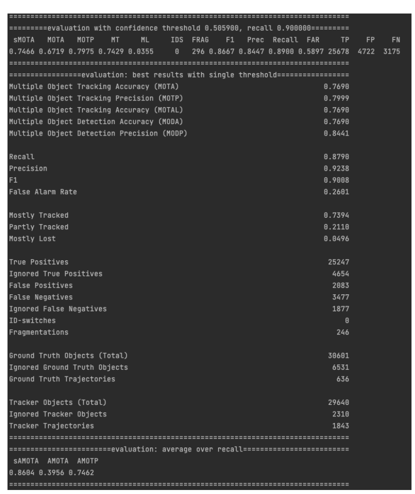
Question 2 (Trivial Update) [1 pt]: You'll notice that even though you've implemented matching, the trackers themselves don't update location. For this question you will implement a trivial update to each tracker in "kalman_filter.py". Given a matched detection z, simply set the associated tracker to have the same bounding box z. In other words, we simply trust the observation 100% when updating the state, neglecting any motion or noise models.

In your pdf please show your code for this part, it should be very simple. Report your evaluation MOTA for this setting (meaning matching=greedy, predict() is unimplemented, and the update is trivial).

Answers: The codes and results are shown as follows, the MOTA is 0.7698. The left image is codes for Trivial Update while the right image is the evaluation result.



```
def update(self, z):
    """
    Add a new measurement (z) to the Kalman filter.
    ----------
    z : (dim_z, 1): array_like measurement for this update.
    """
    z = z.reshape(-1,1)

    # ------------------------- Begin your code here ------------------------------------------
    #TODO: Q2 codes

    self.x[:7] = z

    #TODO: Q4 codes

    # # Kalman gain
    # self.S = dot(self.H, dot(self.Sigma, self.H.T)) + self.R
    # self.SI = np.linalg.inv(self.S)
    # self.K = dot(self.Sigma, dot(self.H.T, self.SI))
    # # measured data
    #
    # # update process and state matrix
    # self.y = z - dot(self.H, self.x)
    # self.x = self.x + dot(self.K, self.y)
    # self.Sigma = dot((eye(self.dim_x) - dot(self.K, self.H)), self.Sigma)
    # self.hits += 1
    # ------------------------- End your code here   ------------------------------------------

    # Leave this at the end, within_range ensures that the angle is between -pi and pi
    self.x[3] = within_range(self.x[3])
    return
```



```
==================================================================================
=========evaluation with confidence threshold 0.505900, recall 0.900000=========
 sMOTA   MOTA   MOTP    MT     ML     IDS  FRAG    F1    Prec  Recall  FAR     TP    FP    FN
0.7466 0.6719 0.7975 0.7429 0.0355     0   296 0.8667 0.8447 0.8900 0.5897 25678  4722  3175
==================================================================================
================evaluation: best results with single threshold=================
Multiple Object Tracking Accuracy (MOTA)                                  0.7690
Multiple Object Tracking Precision (MOTP)                                 0.7999
Multiple Object Tracking Accuracy (MOTAL)                                 0.7690
Multiple Object Detection Accuracy (MODA)                                 0.7690
Multiple Object Detection Precision (MODP)                                0.8441

Recall                                                                    0.8790
Precision                                                                 0.9238
F1                                                                        0.9008
False Alarm Rate                                                          0.2601

Mostly Tracked                                                            0.7394
Partly Tracked                                                            0.2110
Mostly Lost                                                               0.0496

True Positives                                                             25247
Ignored True Positives                                                      4654
False Positives                                                             2083
False Negatives                                                             3477
Ignored False Negatives                                                     1877
ID-switches                                                                    0
Fragmentations                                                               246

Ground Truth Objects (Total)                                               30601
Ignored Ground Truth Objects                                                6531
Ground Truth Trajectories                                                    636

Tracker Objects (Total)                                                    29640
Ignored Tracker Objects                                                     2310
Tracker Trajectories                                                        1843
==================================================================================
======================evaluation: average over recall=======================
 sAMOTA  AMOTA  AMOTP
0.8604 0.3956 0.7462
==================================================================================
```

Question 3 (Kalman Linear Dynamics) [3 pt]: For this part you will fill in define_model() in the class Kalman. The state **x** should consist three dimensional box center, raw angle, three dimensional box size, and finally three dimensional linear velocity (total 10D). The motion model is a constant linear velocity model in 3D. Your model should be linear, meaning x' = x + dx = Ax. In addition you should define the measurement model and measurement uncertainty, meaning H and Sigma and Q. In your pdf please report A, H, Sigma, Q, and R. Explain why each is set the way it is.

Answers: A is the transition matrix which represent the dynamic model for the state. The state x has 10 dimensions: $x, y, z, theta, l, w, h, dx, dy, dz$ For the constant velocity model:

$$x' = x + dx, \quad y' = y + dy, \quad z' = z + dz$$

while all others $theta, l, w, h, dx, dy, dz$ remain the same. In tha case, the A is shown as follows:

```
self.A:  [[1. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

H is the measurement function, the first 7 dimensions of the measurement correspond to the state. The H is shown as follows:

```
self.H:  [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

Sigma is the uncertainty vcovariance, Q is process uncertainty and R is measurement uncertinty, I tune them as follows to get a relatively good performance for multi object tracking

```
self.Sigma[:7, :7] *= 100
# print("self.Sigma: ", self.Sigma)
self.Q[:7, :7] *= 0.01
self.Q[7:, 7:] *= 100
# print("self.Q: ", self.Q)
self.R *= 0.1
self.R[3,3] = 0.01
# print("self.R: ", self.R)
```

Question 4 (Kalman Update) [3 pt]: Now implement a proper Kalman Filter Update step, where you use a matched object detection as a noisy observation for updating the state (see lectures 11-12).

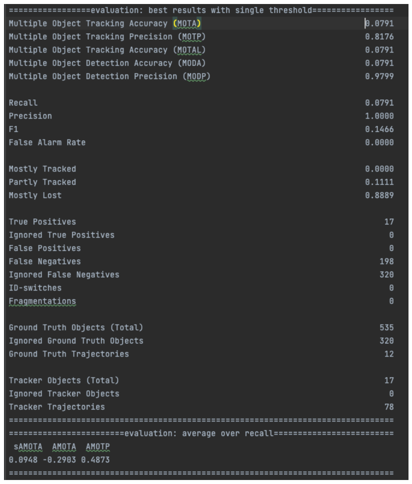
In your pdf please describe the Kalman Filter linear update mathematically and report your evaluation MOTA under this setting (matching=greedy, predict() unimplemented, update implemented).

Answers: The update equations are shown as follows:

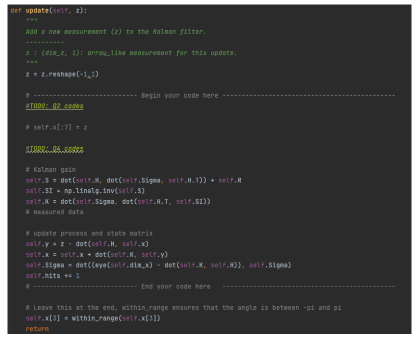
$$S_{t+1} = H\Sigma_{t+1|t}H^T + R \tag{1}$$
$$K_{t+1} = \Sigma_{t+1|t}H^T S_{t+1}^{-1} \tag{2}$$
$$\mu_{t+1} = H\mu_{t+1|t} \tag{3}$$
$$\mu_{t+1|t+1} = \mu_{t+1|t} + K_{t+1}(z_{t+1} - \mu_{t+1}) \tag{4}$$
$$\Sigma_{t+1|t+1} = \Sigma_{t+1|t} - K_{t+1}H\Sigma_{t+1|t} \tag{5}$$

where S_{t+1} is the innovation covariance, K_{t+1} is the Kalman gain, $\Sigma_{t+1|t}$ is the predicted error covariance, $\mu_{z,t+1}$ is the predicted measurement, $\mu_{t+1|t}$ is the predicted state, z_{t+1} is the actual measurement, R is the measurement noise covariance, and H is the measurement matrix.

The evaluation result and codes are shown as follows, MOTA is 0.0791:



```
================evaluation: best results with single threshold================
Multiple Object Tracking Accuracy (MOTA)                             0.0791
Multiple Object Tracking Precision (MOTP)                            0.8176
Multiple Object Tracking Accuracy (MOTAL)                            0.0791
Multiple Object Detection Accuracy (MODA)                            0.0791
Multiple Object Detection Precision (MODP)                           0.9799

Recall                                                               0.0791
Precision                                                            1.0000
F1                                                                   0.1466
False Alarm Rate                                                     0.0000

Mostly Tracked                                                       0.0000
Partly Tracked                                                       0.1111
Mostly Lost                                                          0.8889

True Positives                                                           17
Ignored True Positives                                                    0
False Positives                                                           0
False Negatives                                                         198
Ignored False Negatives                                                 320
ID-switches                                                               0
Fragmentations                                                            0

Ground Truth Objects (Total)                                            535
Ignored Ground Truth Objects                                            320
Ground Truth Trajectories                                                12

Tracker Objects (Total)                                                  17
Ignored Tracker Objects                                                   0
Tracker Trajectories                                                     78
=============================================================================
======================evaluation: average over recall======================
 sAMOTA  AMOTA  AMOTP
0.0948 -0.2903 0.4873
=============================================================================
```



```
def update(self, z):
    """
    Add a new measurement (z) to the Kalman filter.
    ----------
    z : (dim_z, 1): array_like measurement for this update.
    """
    z = z.reshape(-1,1)

    # -------------------------- Begin your code here ------------------------------------------
    #TODO: Q2 codes

    # self.x[:7] = z

    #TODO: Q4 codes

    # Kalman gain
    self.S = dot(self.H, dot(self.Sigma, self.H.T)) + self.R
    self.SI = np.linalg.inv(self.S)
    self.K = dot(self.Sigma, dot(self.H.T, self.SI))
    # measured data

    # update process and state matrix
    self.y = z - dot(self.H, self.x)
    self.x = self.x + dot(self.K, self.y)
    self.Sigma = dot((eye(self.dim_x) - dot(self.K, self.H)), self.Sigma)
    self.hits += 1
    # -------------------------- End your code here   ------------------------------------------

    # Leave this at the end, within_range ensures that the angle is between -pi and pi
    self.x[3] = within_range(self.x[3])
    return
```

Question 5 (Kalman Predict) [2 pt]: Up until now, each frame the detections were compared to each tracker, and then matched trackers were updated. But our matching is poor because the detections and trackers do not overlap (they are one frame apart). In this question you will implement the Kalman Filter Predict step, where you forward propagate the state according to the dynamics model you defined earlier.

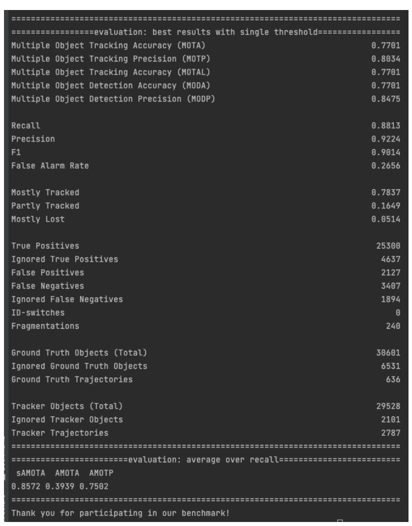
In your pdf please describe the predict step, and report your evaluation MOTA under this setting (matching=greedy, predict and update both implemented).

Answers: The predict function is shown as follows:

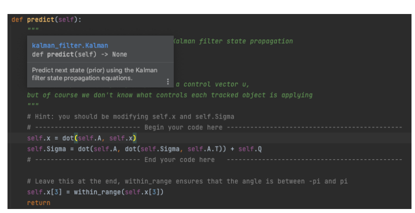
$$\mu_{t+1|t} = A\mu_{t|t} + B\underline{u}_t \tag{6}$$

$$\Sigma_{t+1|t} = A\Sigma_{t|t}A^T + GQG^T \tag{7}$$

The evaluation result and codes are shown as follows, MOTA is 0.7701:



```
=============================================================================
================evaluation: best results with single threshold================
Multiple Object Tracking Accuracy (MOTA)                             0.7701
Multiple Object Tracking Precision (MOTP)                            0.8034
Multiple Object Tracking Accuracy (MOTAL)                            0.7701
Multiple Object Detection Accuracy (MODA)                            0.7701
Multiple Object Detection Precision (MODP)                           0.8475

Recall                                                               0.8813
Precision                                                            0.9224
F1                                                                   0.9014
False Alarm Rate                                                     0.2656

Mostly Tracked                                                       0.7837
Partly Tracked                                                       0.1649
Mostly Lost                                                          0.0514

True Positives                                                        25300
Ignored True Positives                                                 4637
False Positives                                                        2127
False Negatives                                                        3407
Ignored False Negatives                                                1894
ID-switches                                                               0
Fragmentations                                                          240

Ground Truth Objects (Total)                                          30601
Ignored Ground Truth Objects                                           6531
Ground Truth Trajectories                                               636

Tracker Objects (Total)                                               29528
Ignored Tracker Objects                                                2101
Tracker Trajectories                                                   2787
=============================================================================
======================evaluation: average over recall======================
 sAMOTA  AMOTA  AMOTP
0.8572 0.3939 0.7502
=============================================================================
Thank you for participating in our benchmark!
```



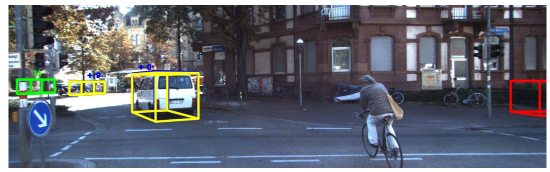
```
def predict(self):
    """
    kalman_filter.Kalman
    def predict(self) -> None
    Predict next state (prior) using the Kalman
    filter state propagation equations.
                                            Kalman filter state propagation
                                            a control vector u,
    but of course we don't know what controls each tracked object is applying
    """
    # Hint: you should be modifying self.x and self.Sigma
    # -------------------------- Begin your code here ------------------------------------------
    self.x = dot(self.A, self.x)
    self.Sigma = dot(self.A, dot(self.Sigma, self.A.T)) + self.Q
    # -------------------------- End your code here   ------------------------------------------

    # Leave this at the end, within_range ensures that the angle is between -pi and pi
    self.x[3] = within_range(self.x[3])
    return
```

Question 6 (Final Visualization) [1 pt]: Please visualize some results from your final code. Pick at least 4 consecutive frames from sequence 0000. For each frame visualize all in one image:

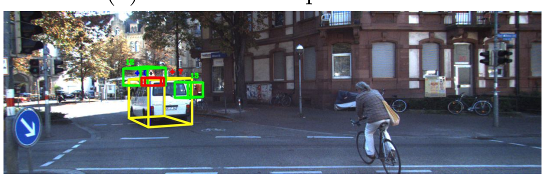
- Show birthed trackers in green
- Show dead trackers in red
- For the rest of the trackers (these were matched), show each before and after update. Show their corresponding detections as well. Color the trackers in blue and detections in yellow. Add text above each tracker with its ID and the text "-" for before update or "+" for after update.

Answers: As shown in the follows BOX9 appears at frame 5 and track as visualized after update in frame 6 and disappear at frame 9. Meanwhile, BOX0 is tracking steady in the whole process.

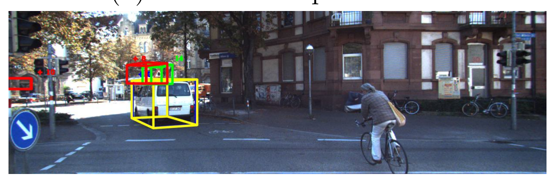
(a) frame 5 in sequence 0000



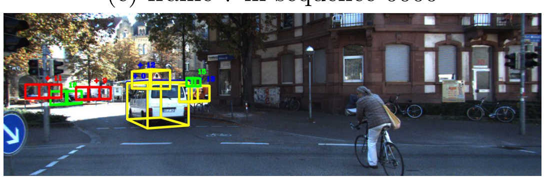
(b) frame 6 in sequence 0000



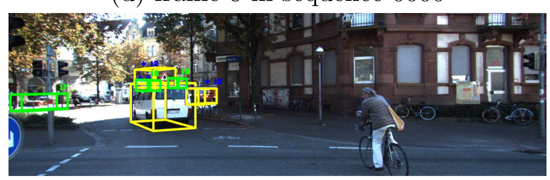
(c) frame 7 in sequence 0000



(d) frame 8 in sequence 0000



(e) frame 9 in sequence 0000



(f) frame 10 in sequence 0000

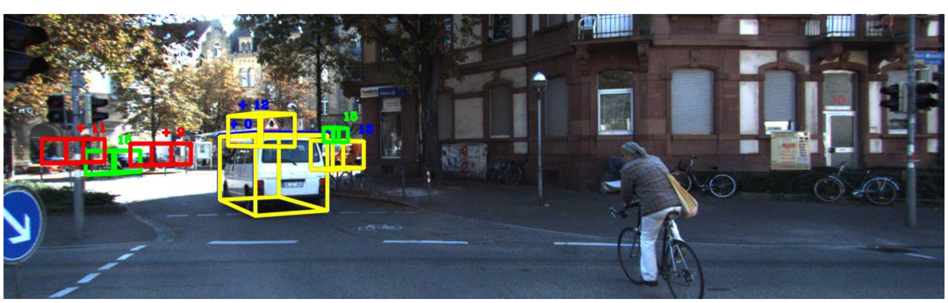
Question 7 (Analysis) [1 pt]: Please run the run `python evaluate.py` and report your MOTA, TPs, ID switches, FRAGs and FPs at the best threshold. Please also visualize at least two failure cases and explain the reason. Please discuss what can be done to avoid these failures.

Answers: MOTA: 0.7701, TPs: 25300, ID switches: 0, FRAGs: 240, FPs 2127, the best threshold is -0.2.

```
=================evaluation: best results with single threshold=================
Multiple Object Tracking Accuracy (MOTA)                                 0.7701
Multiple Object Tracking Precision (MOTP)                                0.8034
Multiple Object Tracking Accuracy (MOTAL)                                0.7701
Multiple Object Detection Accuracy (MODA)                                0.7701
Multiple Object Detection Precision (MODP)                               0.8475

Recall                                                                   0.8813
Precision                                                                0.9224
F1                                                                       0.9014
False Alarm Rate                                                         0.2656

Mostly Tracked                                                           0.7837
Partly Tracked                                                           0.1649
Mostly Lost                                                              0.0514

True Positives                                                            25300
Ignored True Positives                                                     4637
False Positives                                                            2127
False Negatives                                                            3407
Ignored False Negatives                                                    1894
ID-switches                                                                   0
Fragmentations                                                              240

Ground Truth Objects (Total)                                              30601
Ignored Ground Truth Objects                                               6531
Ground Truth Trajectories                                                   636

Tracker Objects (Total)                                                   29528
Ignored Tracker Objects                                                    2101
Tracker Trajectories                                                       2787
================================================================================
=======================evaluation: average over recall=========================
 sAMOTA  AMOTA  AMOTP
0.8572 0.3939 0.7502
================================================================================
```

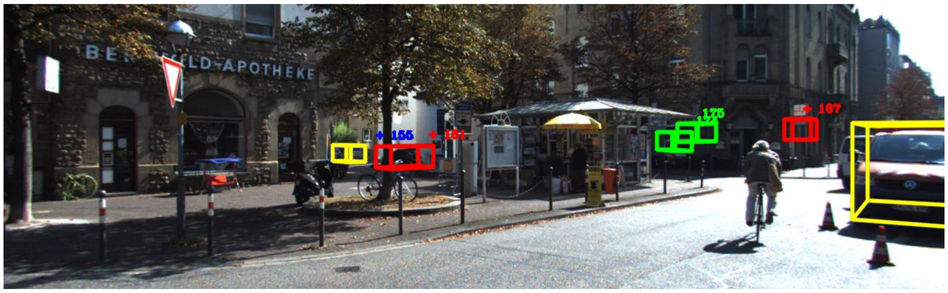
Failure case 1: The detector detects too many bounding boxes that are not targets. This will increase the computational complexity of the system.

Like what I showed below in frame 9 in sequence 0000, there are too many detections in the frame which are not the traget, this is caused by the detectio error of detector network model. If the model is overfitted, we can use more training data or use regularization techniques to prevent it.



Failure case 2: When the tracking object is blocked by other item, the tracking box disappear. This will caused the discontinuity of tracking.

Like what I showed below in frame 82 in sequence 0000, the tracker of the white car(box 0) disappears after white car is blocking. To solve this problem, we can add a estimator for object blocking by other object. this estimator is built based on the motion model of the tracking target.



Bonus Question (Option 1: Better Matching) [3 pt Bonus]: Improve your matching algorithm from greedy to the Hungarian algorithm. You must implement it yourself. Alternatively improve your matching by training a neural network to perform matching based on more than just spatial proximity. Report the tracking performance and compare it against the IOU-based greedy matching through both qualitative and quantitative results. Do you see a performance improvement? Discuss the comparison study results.

Bonus Question (Option 2: Better Dynamics) [3 pt Bonus]: Make your Kalman filter into an extended Kalman filter and implement a bicycle model for dynamics instead of linear. `https://www.coursera.org/lecture/intro-self-driving-cars/lesson-2-the-kinematic-bicycle-model-Bi8yE`. Report the tracking performance and compare it against linear velocity model through both qualitative and quantitative results. Do you see a performance improvement? Discuss the comparison study results.