CS 498: Assignment 5: 3D Multi Object Tracking

Due on Monday, May 2, 2022

April 17, 2022

Submission

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: JBXJVZ). We recommend you add your answers to the latex template files we provided. More details on what to report are in the provided code.

Reminder: please put your name and netid in your pdf.

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 main methods 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, but only 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`

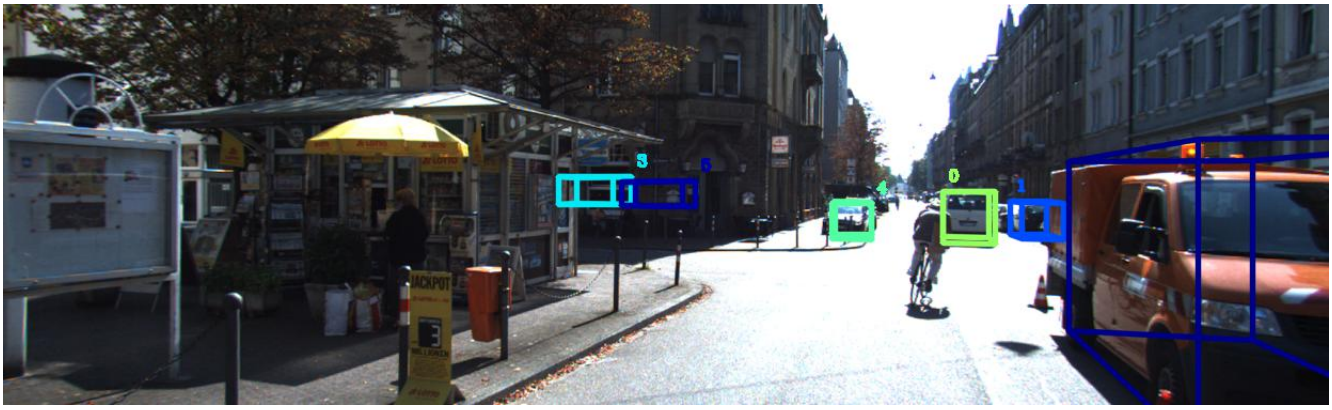

Figure 1: Sequence 0000 — Frame 100

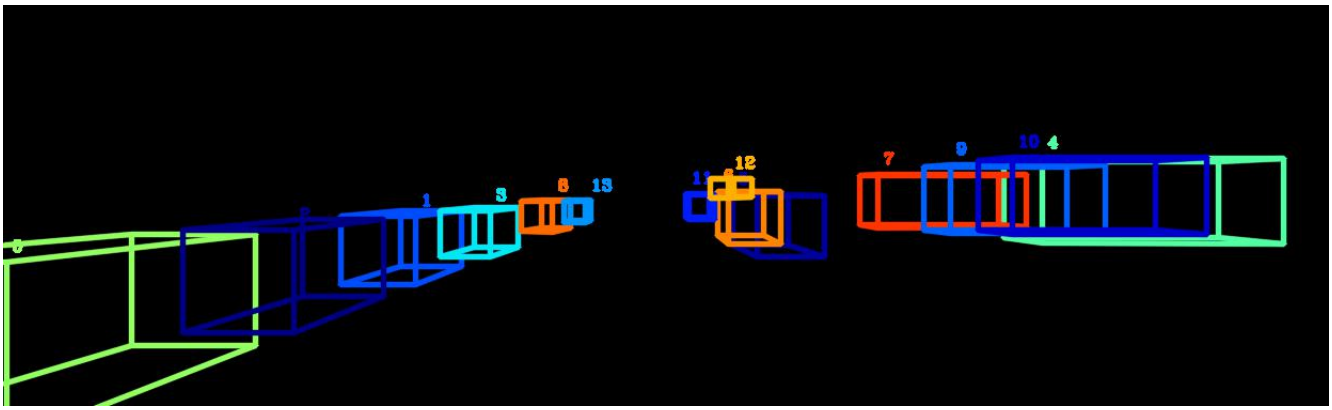

Figure 2: Sequence 0001 — Frame 100

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 more than the provided threshold, do not consider the pair a match.

Notice we provide you with "`iou(box_a, box_b)`" to compute similarity between 3D detected regions.

Answer: To implement greedy matching, I first calculate the pair-wise IoU between detections and trackers, and mask out all the pairs that do not satisfy our requirements for IoU. Then, for each detection, I greedily select the best tracker in terms of IoU. If all the candidate trackers for a detection are masked out, I will simply skip that detection. At last, I calculate all the unmatched detections and trackers. Initially, there might be no matched pairs. I also made sure to handle this edge case. Please see the implementation below.

```
def data_association(dets: List[Box3D], trks: List[Box3D], threshold=-0.2, algm='greedy'):
    if len(dets) == 0:
        return np.array([]), np.array([]), np.arange(len(trks))
    elif len(trks) == 0:
        return np.array([]), np.arange(len(dets)), np.array([])

    matches = []
    pairwise_iou = np.vectorize(lambda x, y: iou(dets[x], trks[y]))(
```

```
        np.arange(len(dets))[:, np.newaxis], np.arange(len(trks))
    )
    pairwise_iou[pairwise_iou < threshold] = np.nan
    for i, match in enumerate(pairwise_iou):
        if np.all(np.isnan(match)):
            # unmatched_dets.append(i)
            continue
        trk_match = np.nanargmax(match)
        matches.append([i, trk_match])
        pairwise_iou[:, trk_match] = np.nan
    matches = np.array(matches)
    if len(matches) == 0:
        unmatched_dets = np.arange(len(dets))
        unmatched_trks = np.arange(len(trks))
    else:
        unmatched_dets = np.setdiff1d(np.arange(len(dets)), matches[:, 0])
        unmatched_trks = np.setdiff1d(np.arange(len(trks)), matches[:, 1])

    return matches, unmatched_dets, unmatched_trks
```

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).

Answer: My implementation for the trivial update is as below.

```
def update(self, z):
    z = z.reshape(-1, 1)

    self.x[:self.dim_z] = z
    self.x[self.dim_z:] = 0

    self.x[3] = within_range(self.x[3])
```

My best MOTA is 0.7690 with trivial update. The evaluation sAMOTA is 0.8604 and the AMOTA is 0.3956. The detailed evaluation results are listed below.

```
=================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 $\boldsymbol{x}' + d\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}$. In addition you should define the measurement model and measurement uncertainty, meaning $\boldsymbol{H}$ and $\boldsymbol{\Sigma}$ and $\boldsymbol{Q}$. In your pdf please report $\boldsymbol{A}$, $\boldsymbol{H}$, $\boldsymbol{\Sigma}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$. Explain why each is set the way it is.

Answer: In the linear motion model, I initialize $\boldsymbol{A}$ as an identity and then set the values at $(0, 7)$, $(1, 8)$, and $(2, 9)$ to 1 because we also assume constant linear velocity $\boldsymbol{x}' + d\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}$ in the motion model. Defining the state transition matrix in this way will update the current positions based on velocities. We initialize $\boldsymbol{Q}$ as a $n_x \times n_x$ diagonal matrix with 5's being diagonal elements. Since the linear motion model is relartively naive compared to more sophisticated bicycle models, we assume high dynamic model noise and low observation noise because I trust the measurements from object detectors more than the linear motion model.

In the linear measurement model, I initialize $\boldsymbol{H}$ as a $n_z \times n_x$ matrix with 1's at the first 7 diagonal elements and 0's otherwise, since we can only observe the positions of bounding boxes and we can't directly measure velocity information. I initialize the measurement uncertainty matrix $\boldsymbol{R}$ as a diagonal matrix with 1's being diagonal elements because the object detector used in this MP seems to predict relatively reliable bounding boxes and we can trust the measurements more.

Finally, I initialize the estimate uncertainty matrix $\boldsymbol{\Sigma}$ as a $n_x \times n_x$ diagonal matrix with 10 being

diagonal elements because the initial estimation uncertainty could be very high due to the lack of initial measurements.

```
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
[[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
[[10.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 10.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 10.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 10.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 10.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 10.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. 10.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 10.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 10.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 10.]]
Q
[[5. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 5. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 5. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 5. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 5. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 5. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 5. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 5. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 5. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 5.]]
R
[[1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0.]
```

```
[0. 0. 0. 1. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0.]
[0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 1.]]
```

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 lecture 10-12 for more details.

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).

Answer: In the update step, I first calculate and update the Kalman gain,

$$\boldsymbol{K}_{t+1} = \boldsymbol{\Sigma}_{t+1|t}\boldsymbol{H}^\top(\boldsymbol{H}\boldsymbol{\Sigma}_{t+1|t}\boldsymbol{H}^\top + \boldsymbol{R})^{-1}$$

where $\boldsymbol{H}$ is the observation matrix, $\boldsymbol{R}$ is the measurement uncertainty, and $\boldsymbol{\Sigma}_{t+1|t}$ is the estimation uncertainty.

Then, I update the current estimation of the states based on new measurement,

$$\boldsymbol{\mu}_{t+1|t+1} = \boldsymbol{\mu}_{t+1|t} + \boldsymbol{K}_{t+1}\left(\boldsymbol{z}_{t+1} - \boldsymbol{H}\boldsymbol{\mu}_{z_{t+1|t}}\right)$$

where $\boldsymbol{z}_{t+1}$ is the new measurement and $\boldsymbol{K}_{t+1}$ is the Kalman gain we just updated.

At last, we update the estimate uncertainty by

$$\boldsymbol{\Sigma}_{t+1|t+1} = \boldsymbol{\Sigma}_{t+1|t} - \boldsymbol{K}_{t+1}\boldsymbol{H}\boldsymbol{\Sigma}_{t+1|t}.$$

Implementing update step only without the prediction step yields a very bad performance with 0.0059 sAMOTA and -0.0141 AMOTA in the final evaluation. However, this is as expected because our motion model is simply adding noise without proper prediction step. Please see the detailed results below.

```
==================evaluation: best results with single threshold=================
Multiple Object Tracking Accuracy (MOTA)                              0.0225
Multiple Object Tracking Precision (MOTP)                             0.7035
Multiple Object Tracking Accuracy (MOTAL)                             0.0225
Multiple Object Detection Accuracy (MODA)                             0.0225
Multiple Object Detection Precision (MODP)                            0.8642

Recall                                                                0.4359
Precision                                                             0.5691
F1                                                                    0.4937
False Alarm Rate                                                      1.0845

Mostly Tracked                                                        0.3245
Partly Tracked                                                        0.2057
Mostly Lost                                                           0.4699

True Positives                                                         11470
Ignored True Positives                                                  2244
False Positives                                                         8685
False Negatives                                                        14844
```

```
Ignored False Negatives                                                 4287
ID-switches                                                                0
Fragmentations                                                           273

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

Tracker Objects (Total)                                                21890
Ignored Tracker Objects                                                 1735
Tracker Trajectories                                                     2669
================================================================================
=========================evaluation: average over recall=========================
 sAMOTA  AMOTA  AMOTP
0.0059 -0.0141 0.3692
================================================================================
```

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).

Answer: In the prediction step, I first extrapolate the current states by

$$\boldsymbol{\mu}_{t+1|t} = \boldsymbol{A}\boldsymbol{\mu}_{t|t},$$

where $\boldsymbol{A}$ is the state transition matrix.

Then, I extrapolate the estimation uncertainty by

$$\boldsymbol{\Sigma}_{t+1|t} = \boldsymbol{A}\boldsymbol{\Sigma}_{t|t}\boldsymbol{A} + \boldsymbol{Q}.$$

Note that we don't have control input $\boldsymbol{u}_t$ and control matrix $\boldsymbol{G}$.

My best MOTA is 0.7756. The evaluation sAMOTA is 0.8609 and the AMOTA is 0.3963. Please see the detailed results below.

```
=================evaluation: best results with single threshold================
Multiple Object Tracking Accuracy (MOTA)                               0.7756
Multiple Object Tracking Precision (MOTP)                              0.8057
Multiple Object Tracking Accuracy (MOTAL)                              0.7756
Multiple Object Detection Accuracy (MODA)                              0.7756
Multiple Object Detection Precision (MODP)                             0.8472

Recall                                                                 0.8799
Precision                                                              0.9283
F1                                                                     0.9035
False Alarm Rate                                                       0.2438
```

```
Mostly Tracked                                                    0.7323
Partly Tracked                                                    0.2163
Mostly Lost                                                       0.0514

True Positives                                                   25272
Ignored True Positives                                            4651
False Positives                                                   1952
False Negatives                                                   3449
Ignored False Negatives                                           1880
ID-switches                                                          0
Fragmentations                                                     218

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

Tracker Objects (Total)                                          29264
Ignored Tracker Objects                                           2040
Tracker Trajectories                                              1835
================================================================================
========================evaluation: average over recall========================
 sAMOTA  AMOTA  AMOTP
0.8609 0.3963 0.7524
================================================================================
```

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 predict and after update. Show their corresponding detections. Color the trackers in blue and detections in yellow. Add text above each tracker with its ID and the text "-" for before predict or "+" for after update.

Answer: Please see the visualizations in the next page.

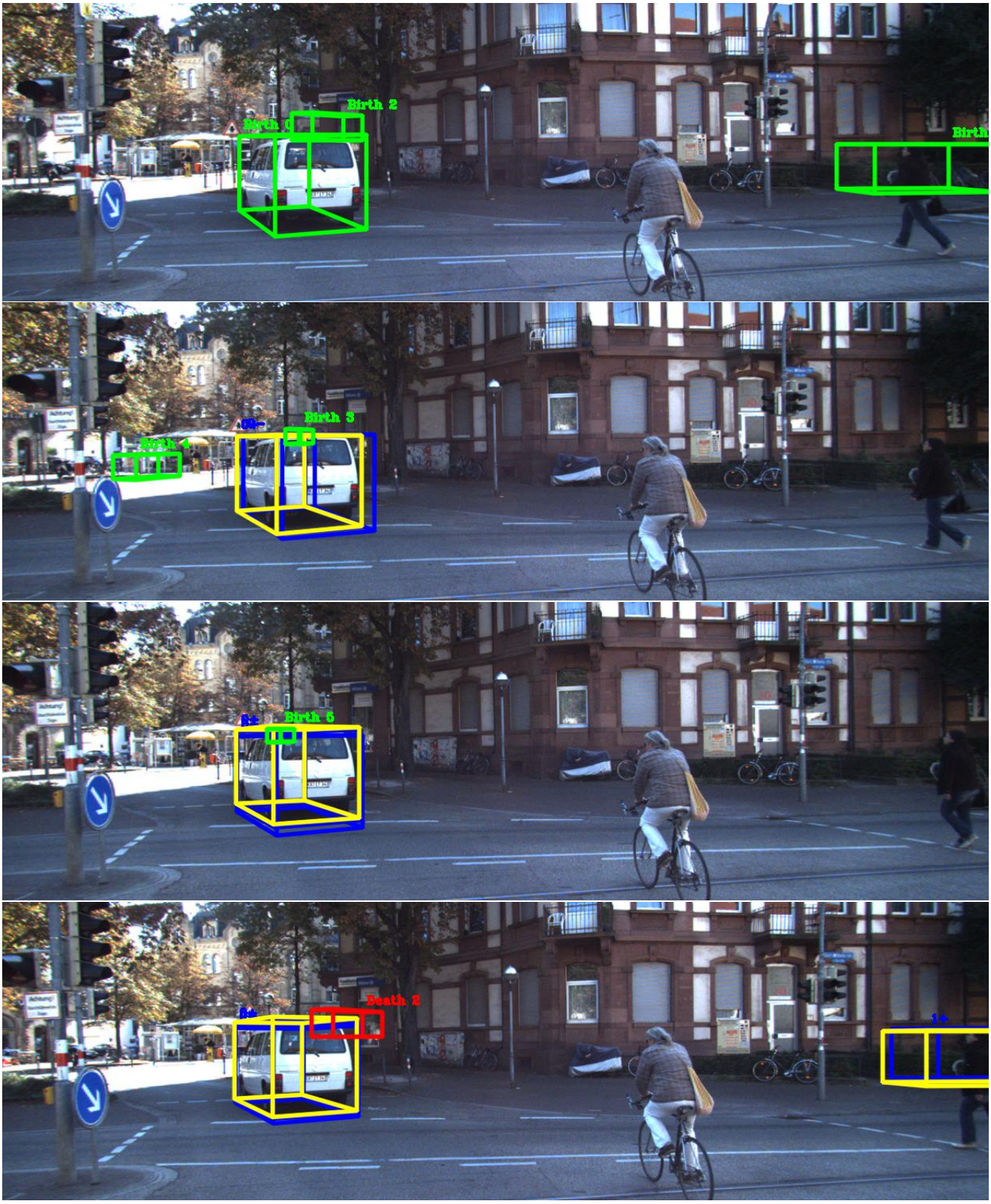

Figure 3: Frame 0 - 3 from Sequence 0000

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.

```
===============evaluation with confidence threshold 2.891237, recall 0.875000=============
 sMOTA    MOTA    MOTP     MT     ML     IDS  FRAG    F1    Prec  Recall  FAR     TP    FP    FN
0.8864 0.7756 0.8057 0.7323 0.0514     0   218 0.9035 0.9283 0.8799 0.2438 25272  1952  3449
=========================================================================================
```

Failure Case 1: Please pay attention to the boxed tracker in the image below. The estimated yaw of the vehicle is not quite correct. The vehicle is apparently parked along the sidewalk, but the blue tracker indicates that is is tilted. This is likely due to the naive linear motion model. We can use more sophisticated vehicle model like bicycle model as our model model to better estimate the vehicle's pose.

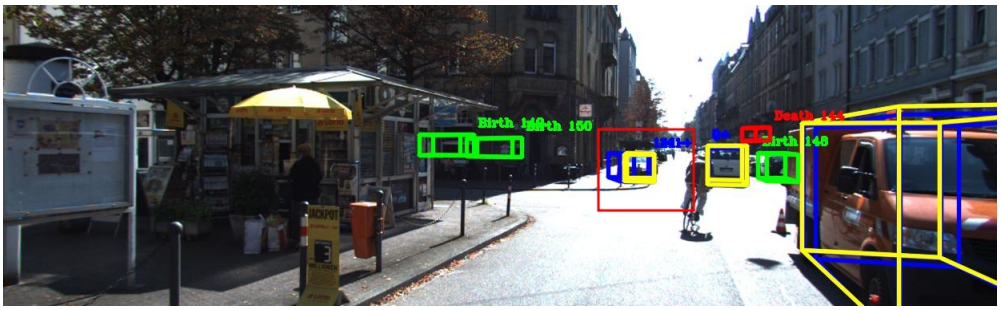

Figure 4: Failure Case 1

Failure Case 2: Please pay attention to the erroneously created trackers. In the first image, the grass is incorrectly tracked possibly due to detection or matching error. In the second image, some non-vehicle objects on the sidewalk are also incorrectly tracked. We can use better detector and better matching algorithm to better deal with these issues.

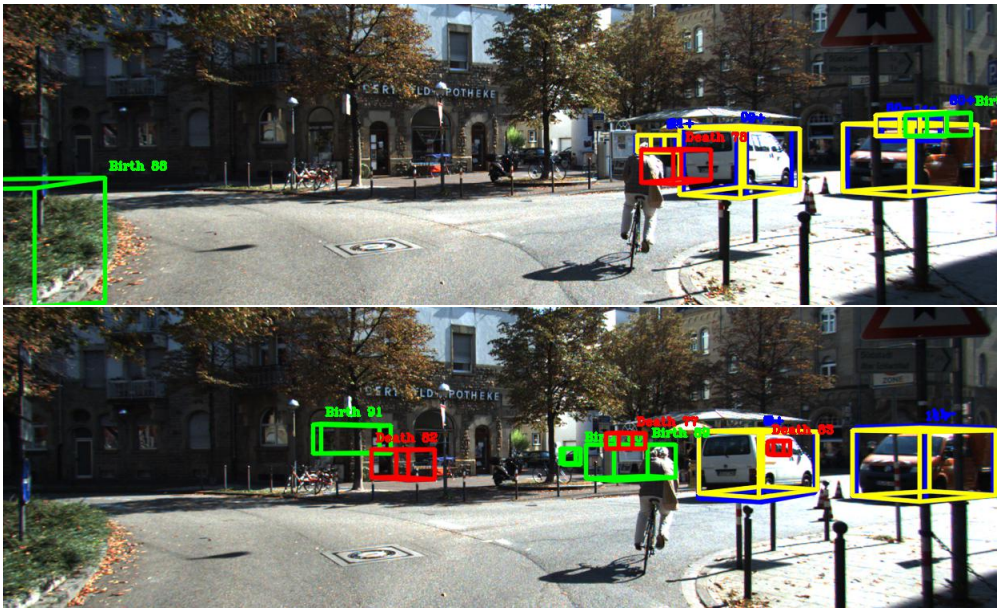

Figure 5: Failure Case 2

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 bycicle 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.