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`

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.

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).

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.

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).

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).

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.

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.

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.