3D OBJECT TRACKING

Project – 3 of Sensor Fusion Nano Degree, Udacity.

FP.0

This document is submitted to be considered as the write-up for this project.

FP.1 Match 3D Objects

In this task, each bounding box in the previous frame is matched with a bounding box in the current frame that has the highest number of Keypoint correspondences.

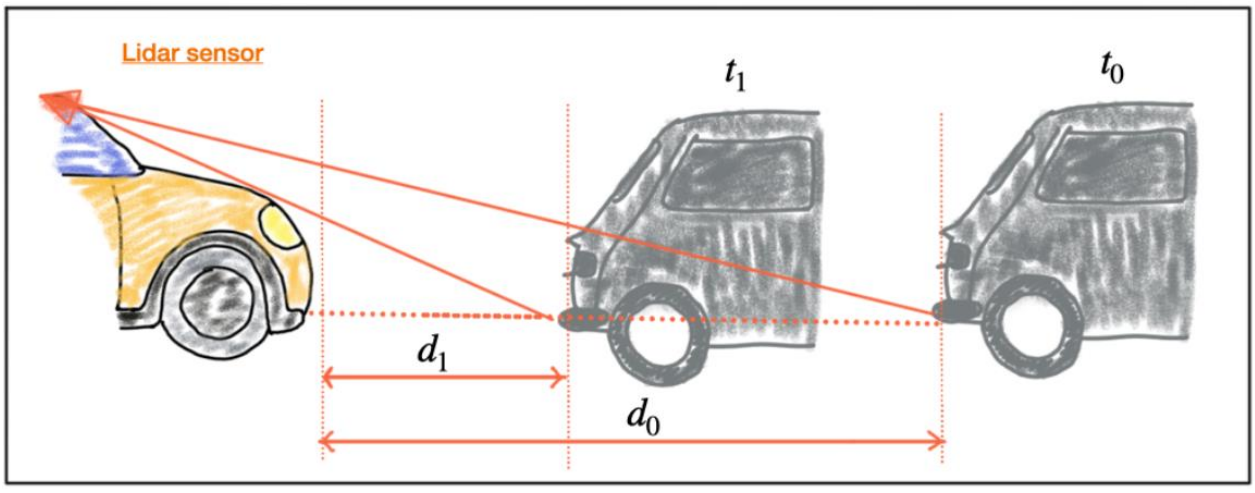
- A two dimensional vector is initialized as a counter. This 2D vector can be assumed as a 2D matrix with rows representing each bounding box of the previous frame and the columns represent the bounding boxes of the current frame.
- For every match, the current and the previous bounding boxes are identified and the occurrence is recorded in the 2D-counter previously initialized. Based on the counter values, for every bounding box in the previous frame, a bounding box in the current frame is assigned as a match.

Note: In some instances, multiple bounding boxes in the previous frame have common match in the current frame. This could be because, over the frames, multiple objects could be merging into one object or overlap each other. In order to overcome this problem, a morphology technique called “Erosion” in OpenCV can be implemented to separate two objects in single frame. However, this problem is beyond the scope of this task.

FP.2 Compute Lidar-based TTC

The computeTTCLidar function is implemented in this task. This function takes the Lidar-points that belong to the current and previous bounding boxes and returns the time to collision (TTC).

LiDAR is an active sensor that generates a point cloud of the environment in its visual space. Though LiDAR is highly reliable, there is still an insignificant possibility for noise in the point cloud. The noise could be in the form of points that are very close to the sensor, which would further result in faulty TTC values.



Hence, to make the function robust towards noise points, the LiDAR x-values are arranged in the ascending order and the point that stands at the 30th percentile is considered in both the current and previous frames. After multiple tuning sessions, the **30th** percentile is found to be the optimal percentile for a consistent TTC estimate.

Further, based on the following formula (3), the TTC value is computed.

$$(1) \quad d(t+\Delta t) = d(t) - v_0 \cdot \Delta t$$

$$(2) \quad v_0 = \frac{d(t) - d(t+\Delta t)}{\Delta t} = \frac{d_0 - d_1}{\Delta t}$$

$$(3) \quad TTC = \frac{d_1}{v_0} = \frac{d_1 \cdot \Delta t}{d_0 - d_1}$$

FP.3 Associate Keypoint Correspondences with Bounding Boxes

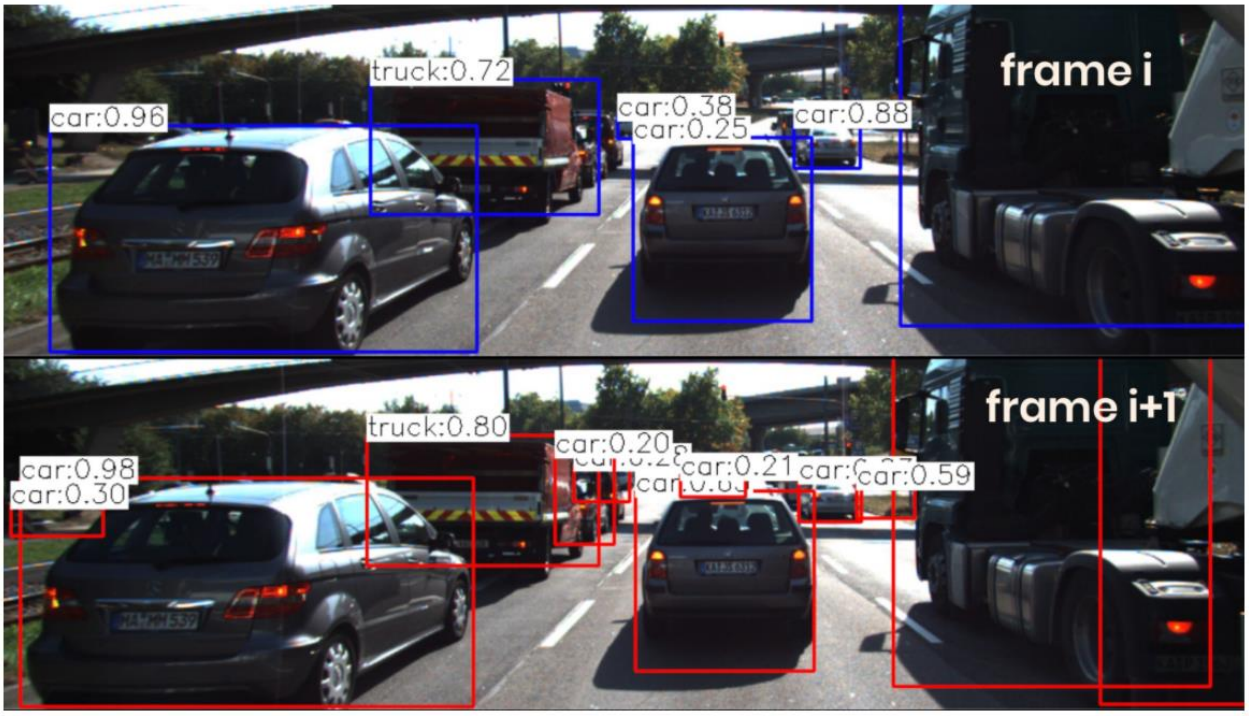
Given all the Keypoint matches and a bounding box in the current frame, the function clusterKptMatchesWithROI identifies the keypoint matches that belong to the given bounding box and adds them to the kptMatches variable of the bounding box. Also, some matches are filtered to avoid erroneous matches. This is done in following steps:

- It is verified if the current Keypoint of each match is within the region of interest (ROI). Then the match and the Euclidian distance between the current Keypoint and the previous Keypoint inserted as a pair into a multimap. By using the multimap data structure, an additional For loop is avoided to loop over all the matches (within ROI) to calculate the distances, thus reducing the time complexity.
- The matches are filtered based on their Euclidian distances. It is assumed that, over the frames, the keypoints move very smoothly in the image frame space. Thus any large or tiny distances could represent erroneous matches. Assuming that the Euclidian distances form a normal distribution, the mean and standard deviation (sd) are calculated. The matches whose distances are beyond [mean – (1*sd), mean + (1*sd)] are eliminated, which would nearly eliminate 30% of the matches. Thus making the TTC computation robust toward erroneous matches.
- By verifying if Euclidian distance of a match in the multimap is within the desired range, valid matches are added to the kptMatches vector variable of the given bounding box.

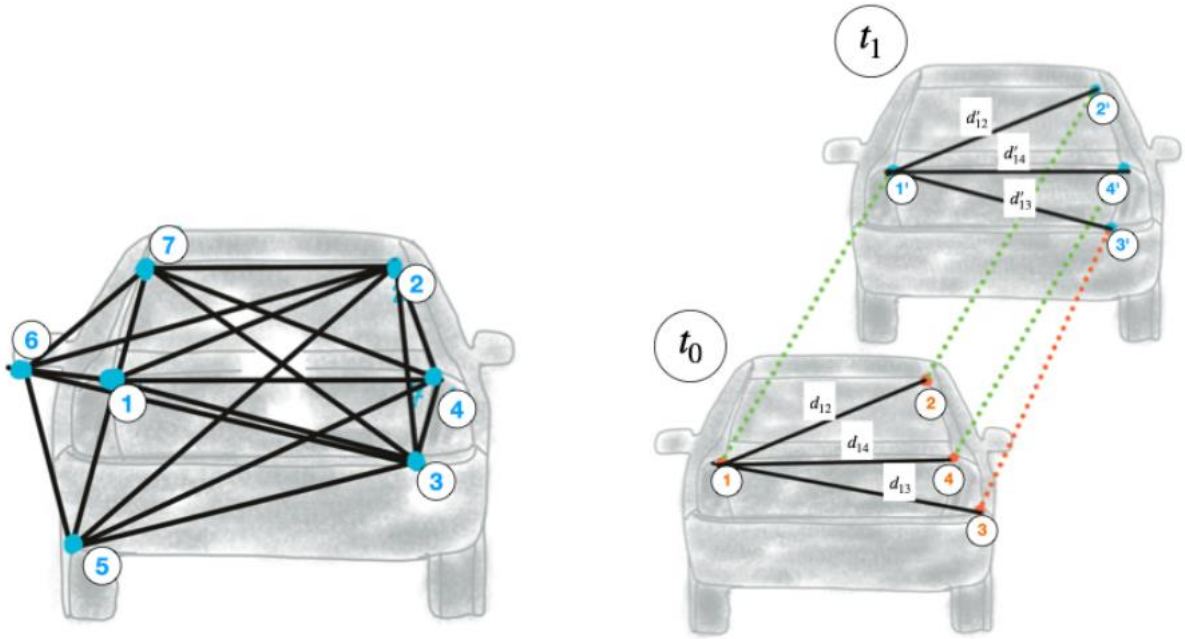
FP.4 Compute Camera-based TTC

This function computes the TTC based on the keypoint information belonging to a bounding box. The simplest way to calculate TTC based on the camera information would be to consider the relationship between the size of the bounding box in the previous and current frame. However, often, bounding

boxes do not accurately represent the size of the object, as they cover a region slightly more than the object itself.



Hence in this task, for every pair of matches, the Euclidian distances between their current keypoints and the previous keypoints is calculated and the distance ratios are stored in a vector.



(a) relative distances between keypoints in an image

(b) selected keypoint distances between successive frames

For a descent estimate of change in the size of the object over the frames, the median distance ratio is considered. The TTC is then computed using the following formula:

$$TTC = \frac{d_1}{v_0} = \frac{-\Delta t}{\left(1 - \frac{h_1}{h_0}\right)}$$

Here, h1/h0 is the median distance ratio as mentioned earlier.

FP.5 Performance Evaluation 1 (TTC-LiDAR)

The TTCs in frames 4, 7 and 18 are way off. The TTC-LiDAR are calculated and curated in the tabular column in TTC Lidar.xlsx file in the "Results" folder.

To calculate the TTC, the LiDAR points are filtered, by ordering the x-distances in ascending order and considering the 30th percentile distance to avoid noise candidates. Hence, the xmin printed in the top-view image of the LiDAR point cloud is different from the filtered-xmin used for the TTC calculation.

There is a steep rise in TTC in frames 4 and 18; The TTC falls suddenly in frame 7. The change in xmin from the previous frames is higher in frames 4, 18 and lower in frame 7 (observe column "Filtered-xmin"). Because a constant velocity model is used, this in consistent change in xmin values give rise to inconsistent TTC estimates. The target vehicle could be accelerating in frames 4, 18 and decelerating in frame 7 which cannot be taken into account by the constant velocity model.

Scope for improvement: A constant acceleration model could potentially improve the quality of TTC estimates. This can be implemented by considering the x distances in the last 2 frames along with the current one.

FP.6 Performance Evaluation (TTC-Camera)

The TTCs are estimated for all the compatible detector + descriptor combinations. SIFT detector + ORB descriptor combination results in openCV error, and the AKAZE descriptor works only with AKAZE keypoints.

All the descriptors with HARRIS detector, and all descriptors, except BRIEF, with ORB detector performed the worst, yielding many +/- infinite TTC numbers. FAST + BRISK, BRISK + BRIEF, AKAZE + BRIEF yielded few negative TTC numbers.

In order to arrive at the three best performers, combinations are eliminated based on the following criteria, in the order mentioned:

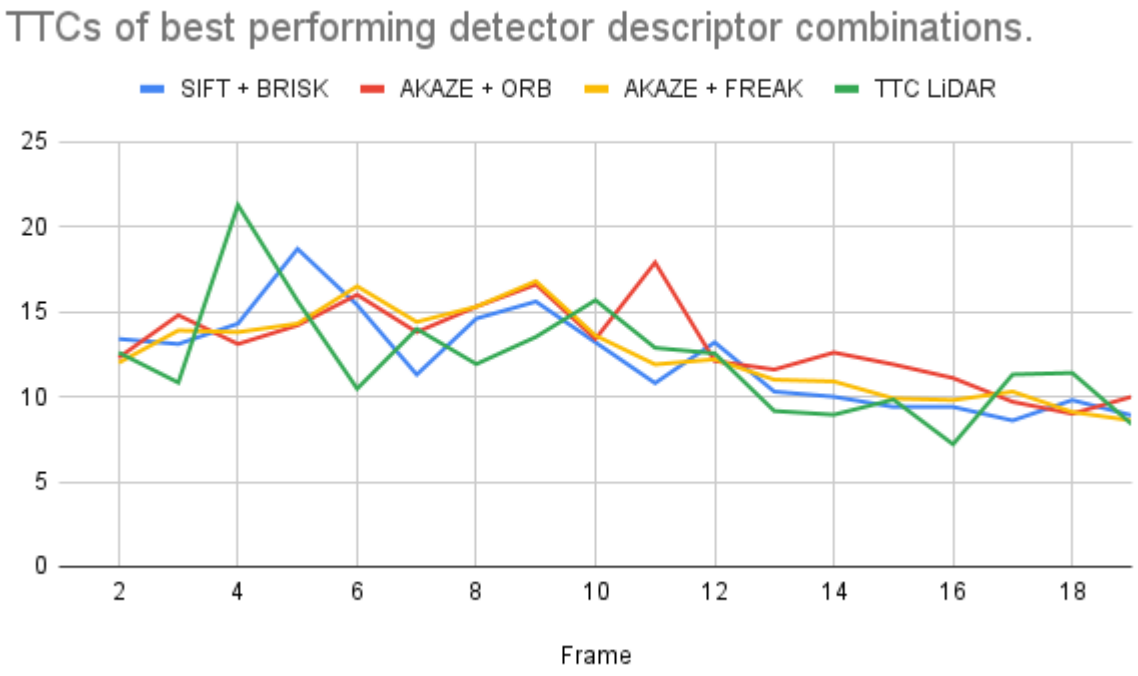
1. All combinations with +/- infinite values.
2. All combinations with - ve numbers.

3. All combinations, whose standard deviation(sd) of TTCs fall beyond the range of (sd_LiDAR ± 0.5 sd_LiDAR).
    a. Note: Since LiDAR TTC is relatively more reliable than camera TTC, it's standard deviation is taken as criteria to eliminate combinations which contain outliers in their TTCs beyond acceptable limits.

Then, three best combinations whose mean is closer to the TTC-LiDAR's mean and whose TTCs are consistent over the frames without weird outliers are selected. The three best combinations, thus, are:

1. SIFT + BRISK
2. AKAZE + ORB
3. AKAZE + FREAK

Please find the corresponding excel file "TTC Camers.xlsx" in results folder.



Initially the TTCs are high. In the second half of the frames, as the ego car gets closer, the TTC reduces. Overall, the TTCs of the camera based estimators follow the trend of TTC-Lidar, except for the TTC-Lidar's peak in frame 4, which is not resembled in the TTC of any of the detector+descriptor combinations. In addition, the Camera based TTCs are slightly smoother than the ones based on Lidar.