Final project Report

Antonino Bongiovanni

FP.1 Match 3D Objects

To implement the method "matchBoundingBoxes", I have to verify if the previous bounding box contains the previous keypoints (`prevFrame.keypoints[match.queryIdx]`), and at the same time, the current bounding box contains the current keypoints. I implemented this logic in the first for loop, exploiting the method `.contains()` of the roi of each bounding box.

After this loop, I obtained a map with a `std::pair<int, int>` as key, containing the IDs of the matched bounding box (`prev_boxID`, `curr_boxID`), and an `int` as value representing the number of good matches between those bounding boxes.

The last loop is necessary in order to obtain the pair (`prev_boxID`, `curr_boxID`) with most matches for each bounding box among the previous bounding boxes.

FP.2 Compute Lidar-based TTC

To evaluate the time-to-collision through the Lidar, I computed the mean of the x values of keypoints in lidarPointsPrev to mitigate the presence of outliers, the same procedure was applied to the keypoints belonging to lidarPointsCurr. Obtained the two values, I used them to compute the TTC.

$$TTC = \frac{minXCurr * dT}{minXPrev - minXCurr}; \tag{1}$$

with:

- $dT = \frac{1}{framerate}$
- minXCurr: mean of the x value of keypoints in lidarPointsCurr
- minXPrev: mean of the x value of keypoints in lidarPointsPrev

FP.3 Associate Keypoint Correspondences with Bounding Boxes

The main goal of this task is to associate keypoints correspondences with bounding boxes, and at the same time removing keeping an eye on the presence of outliers.

First of all, I selected all the keypoints within the bounding box by iterating over all the matches, and then I computed the euclidean distance within the keypoints, storing this information in a vector. Then, I computed the mean and the standard deviation of the distribution of euclidean distance and thresholded the keypoints, keeping those inside the interval $[\mu - 2\sigma, \mu + 2\sigma]$ (to keep the 95% of the data). I believe that this is a robust way of eliminating the randomly occurence of outliers.

Keypoints and matches are then stored inside the data structure BoundingBox.

FP.4 Compute Camera-based TTC

To evaluate the camera-based time-to-collision, I computed the distance ratios between all matched keypoints . Then, I selected the median among the vector of euclidean distance ratios and exploited it in the following formula to obtain the TTC.

$$TTC = -dT/(1 - medianDistRation); \qquad (2)$$

with:

- $dT = \frac{1}{framerate}$
- medianDistRation: median of euclidean distance ratios

FP.5 : Performance Evaluation 1

In the compute TTC with Lidar, there are two ways of computing the closest point exploited then for the calculation. The former consists in estimating the closest point to the lidar frame, this method does not contain any logic to get rid of the outliers. The latter consists in estimating the mean of the points and exploiting it as the closest point. With this logic, we start being robust against outliers.

The robustness of the second alternative (the mean based) is shown by computing the TTC in both cases:

TTC closest point	TTC mean point
12.9722	12.2891
12.264	13.3547
13.9161	16.3845
7.11572	14.0764
16.2511	12.7299
12.4213	13.7511
34.3404	13.7314
9.34376	13.7901
18.1318	12.059
18.0318	11.8642
3.83244	11.9682
-10.8537	9.88711
9.22307	9.42504
10.9678	9.30215
8.09422	8.3212
3.17535	8.89867
-9.99424	11.0301
8.30978	8.53557

As can be seen from the data, the first column not only presents more fluctuating values, but also there are some negative values, which are unfeasible.

FP.6 : Performance Evaluation 2

The results of all the combinations detector / descriptor are showed in the .png files, where it is reported the TTC estimation (s) and the whole computation (ms). From my results, the combinations FAST/ORB and FAST/BRISK are the more performant.