Final Project Report
Camera Lessons
Sensor Fusion Nano degree
Udacity

Ayush Jain
Submission Date: 29/04/2020

FP.0 Final Report

Q. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

Specification Needed The writeup / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled.

Ans: The current document is the writeup requested. Please read through.

FP.1 Match 3D Objects

Q. Implement the method "matchBoundingBoxes", which takes as input both the previous and the current data frames and provides as output the ids of the matched regions of interest (i.e. the boxID property). Matches must be the ones with the highest number of keypoint correspondences.

Specification Needed: Code is functional and returns the specified output, where each bounding box is assigned the match candidate with the highest number of occurrences.

Ans: The **matchBoundingBoxes** function iterates through all points found in the **matches** vector, and determines if the point in question exists in the previous and current frames. If this is true, the count of the occurrence is updated within a temporary vector named **listOfMatches**. Otherwise, the loop continues on to the next point in the **matches** vector. Once all points in the **matches** vector have been compared, another **for** loop is used to iterate through all points in the temporary vector, and store the best points in the **bbMatches** map.

FP.2 Compute Lidar-based TTC

Q. Compute the time-to-collision in second for all matched 3D objects using only Lidar measurements from the matched bounding boxes between current and previous frame.

Specification Needed: Code is functional and returns the specified output. Also, the code is able to deal with outlier Lidar points in a statistically robust way to avoid severe estimation errors.

Ans: The time to collision (TTC) was calculated using the following formula:

```
TTC = (minXCurr * dT) / (averageXPrev - averageXCurr);
```

The **minXCurr** is distance of the closest lidar point from the preceding vehicle. The **averageXCurr** and **averageXPrev** are the average distances of lidar points in the current and preceding frames, respectively.

The **minXCurr** is found using the following code.

```
    for (auto point : lidarPointsCurr)
    {
        const double threshold = 0.75 * averageXCurr ;

        if (point.x > 0.0 && point.x > threshold)
            minXCurr = point.x;
    }
```

The code handles erroneous measurements of lidar points. Especially the coordinate in the x direction. Here, We considered only those lidar points, whose x direction measurement are more than 75% of average x direction measurement.

FP.3 Associate Keypoint Correspondences with Bounding Boxes

Q. Prepare the TTC computation based on camera measurements by associating keypoint correspondences to the bounding boxes which enclose them. All matches which satisfy this condition must be added to a vector in the respective bounding box.

Specification Needed: Code performs as described and adds the keypoint correspondences to the "kptMatches" property of the respective bounding boxes. Also, outlier matches have been removed based on the euclidean distance between them in relation to all the matches in the bounding box.

Ans: The keypoint matches are clustered and associated with bounding boxes in the **clusterKptMatchesWithROI** function. A **for** loop iterates through all points in the **kptMatches** vector, checks if the point exists in the given bounding box, and if so, stores it in a vector. Next, the average distance of all points found in the bounding box is calculated. Finally, another **for** loop again iterates through all the points in the **kptMatches** vector, checks if it's again within the bounding box, and if the current point falls within a distance threshold. If so, the point is stored in that bounding box's keypoint matches.

```
        if (cv::norm(kptCurr.pt - kptPrev.pt) < scaledDistanceMean)
        {
            boundingBox.keypoints.push_back(kptCurr);
            boundingBox.kptMatches.push_back(point);
        }
```

FP.4 Compute Camera-based TTC

Q. Compute the time-to-collision in second for all matched 3D objects using only keypoint correspondences from the matched bounding boxes between current and previous frame.

Specification Needed: Code is functional and returns the specified output. Also, the code is able to deal with outlier correspondences in a statistically robust way to avoid severe estimation errors.

Ans: The formula used in the **computeTTCCamera** function was adapted from the previous lessons. The TTC was calculated by iterating through all points in the **kptMatches** vector, and each of those points to all other points in the same vector using an inner **for** loop.

The formula used to calculate the TTC is as follows:

```
TTC = -dT / (1 - medDistRatio);
```

The median distance was used to remove any outlier influence. In addition, TTC is also calculated using different combinations of detector/descriptors combination.

FP.5 Performance Evaluation 1

Q. Find examples where the TTC estimate of the Lidar sensor does not seem plausible. Describe your observations and provide a sound argumentation why you think this happened.

Specification Needed: Several examples (2-3) have been identified and described in detail. The assertion that the TTC is off has been based on manually estimating the distance to the rear of the preceding vehicle from a top view perspective of the Lidar points.

Ans: In some cases, the TTC Lidar estimation were obviously inaccurate, and this could have occurred due to following reasons.

1. When the number of key points detected by the detector and descriptor combination were very few in number. Thus, due to availability of a very small number of points for the measurements of TTC, the poor accuracy to calculate TTC estimation can be explained. This can be noticed when the Harris detector was used in combination with the other descriptor type.

2. When the number of frames taken into account for each TTC calculation is only 2. In practice, this is unrealitic since it could lead to noisy estimates as small variations in lidar measurements between frames could lead to undue variations in the TTC.

3. Another reason can be that the TTC estimate equation does not take into account the acceleration with respect to the preceding car since the equation is constant- velocity based.

FP.6 Performance Evaluation 2

Q. Run several detector / descriptor combinations and look at the differences in TTC estimation. Find out which methods perform best and also include several examples where camera-based TTC estimation is way off. As with Lidar, describe your observations again and also look into potential reasons.

Specification Needed: All detector / descriptor combinations implemented in previous chapters have been compared with regard to the TTC estimate on a frame-by-frame basis. To facilitate comparison, a spreadsheet and graph should be used to represent the different TTCs.

Ans: A spreadsheet containing the final results can be found here:
report/Ayush_Jain_Final_3D_Camera_Project.csv.

```
IMAGE NO., DETECTOR TYPE, DESCRIPTOR TYPE, TOTAL KEYPOINTS, TTC LIDAR, TTC CAMER
A,
0, SHITOMASI, BRISK, 1723, 6.9181e-310, 6.9527e-310
1, SHITOMASI, BRISK, 1759, 12.3245, 14.1119

0, SHITOMASI, BRIEF, 1723, 6.9181e-310, 6.9527e-310
1, SHITOMASI, BRIEF, 1759, 12.3245, 13.8948

0, SHITOMASI, ORB, 1723, 6.9181e-310, 6.9527e-310
1, SHITOMASI, ORB, 1759, 12.3245, 13.8801

0, SHITOMASI, FREAK, 1723, 6.9181e-310, 6.9527e-310
1, SHITOMASI, FREAK, 1759, 12.3245, 13.7249

0, SHITOMASI, SIFT, 1723, 6.9181e-310, 6.9527e-310
1, SHITOMASI, SIFT, 1759, 12.3245, 14.0746

0, HARRIS, BRISK, 115, 6.9181e-310, 6.9527e-310

0, HARRIS, BRIEF, 115, 6.9181e-310, 6.9527e-310

0, HARRIS, ORB, 115, 6.9181e-310, 6.9527e-310

                                                              25,0-1        3%
```

Figure 1:

Figure 2: Tracking an Object in 3D Space