Udacity Sensor Fusion - Final Project - SFND 3D Object Tracking

Deniz Bardakci

February 2024

1 MP.0 Mid-Term Report

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. This document serves to fullfill this requirement.

2 FP.1 Match 3D Objects

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.

- The matchBoundingBoxes function is implemented under camFusion_Student.cpp in line 287.
- The matchBoundingBoxes function establishes correspondences between bounding boxes across consecutive frames (previous and current) by analyzing keypoint matches from feature matching. It constructs a 2D count matrix to tally how many keypoints from matched features fall within the overlapping bounding boxes between frames. For each match, it identifies which bounding boxes in the previous and current frames contain the keypoints, incrementing the respective counts. After processing all matches, it determines the best match for each bounding box in the previous frame by selecting the current frame's bounding box with the highest count of matched keypoints. This approach ensures that the bounding box relationships are based on the most significant overlaps, facilitating accurate tracking of objects across frames.

3 FP.2 Compute Lidar-based TTC

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

- The computeTTCLidar function is implemented under camFusion_Student.cpp in line 237.
- The computeTTCLidar function is designed to calculate the Time-to-Collision (TTC) using Lidar data by evaluating the change in the position of objects detected in consecutive frames. It operates by first extracting the x-coordinates of Lidar points from two sets of data (representing previous and current frames), which are then used to compute the median distances

for both frames. This median-based approach helps to mitigate the effect of outliers, providing a more robust estimation of the object's distance by focusing on the central tendency of the measured distances rather than their average. A critical step involves calculating the TTC based on these median distances, with the calculation considering the time elapsed between the two measurements (derived from the frame rate) and the relative distance change. If the distance to the object decreases over time, indicating a potential collision, the TTC is calculated using the median distances. However, if the median distance does not decrease (implying any risk of collision or the object moving away), the function returns a NaN value, indicating that the TTC is not calculable under the given conditions. This method ensures the reliability of the TTC estimate in scenarios where Lidar data is used for collision detection or avoidance systems.

4 FP.3 Associate Keypoint Correspondences with Bounding Boxes

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

- The target is achieved by implementing clusterKptMatchesWithROI in line 142 under camFusion_Student.cpp.
- The clusterKptMatchesWithROI function filters keypoint matches between two consecutive frames (previous and current) within a specified region of interest (ROI) defined by a bounding box. It calculates the Euclidean distance between matched keypoints, and then computes the mean distance of all matches to identify and exclude outliers. Matches, where the current keypoint falls within the acceptable distance range from the mean and is contained within the bounding box, are considered valid and retained. The function ultimately updates the bounding box object with these filtered matches and their associated key points, ensuring that only relevant and reliable matches are used for subsequent processing or analysis.

5 FP.4 Compute Camera-based TTC

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

- The target is achieved by implementing computeTTCCamera function under camFusion_Student.cpp in line 185.
- The computeTTCCamera function calculates the Time-to-Collision (TTC) for a camera-based monitoring system by analyzing the distances between matched key points in consecutive frames (previous and current). It first computes the ratio of distances between each pair of matched key points across the two frames, filtering out those with a distance smaller than a predefined threshold to avoid division by zero and mitigate the impact of noise. The function then determines the median of these distance ratios to estimate the relative speed of the observed object robustly, minimizing the influence of outliers. Finally, using the median

distance ratio and the camera's frame rate, it calculates the TTC, which estimates how long it will take for a collision to occur at the current relative velocity, returning NAN if no valid ratios indicate an undefined TTC.

6 FP.5 Performance Evaluation 1

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.

- Since Lidar-based TTC calculation does not depend on the keypoint extraction/matching process lidar-based TTC computation the same for all descriptor/detector pairs and the results have been shared as Table: 1.
- In this table, image 4 and image 5 might be the questionable results for lidar-based TTC estimation durations. As mentioned in Section 3, the median-based outlier method has been utilized. A more resilient way of filtering out outlier Lidar points would be helpful to mitigate the faulty Lidar-based estimation in a more performance and accuracy way. According to my search on Google, other candidates for the outlier rejection method could be the Interquartile Range (IQR), Z-Score Filtering, or Density-Based Spatial Clustering of Applications with Noise (DBSCAN).
- Other than the mentioned issue, Lidar-based TTC computations are much more reliable than Camera-based approaches for the TTC estimations.
- As a final comment, I believe these performance results are hard to applicable in dynamic environments especially when the ego vehicle is performing at high speeds like 90 km/h.

7 FP.6 Performance Evaluation 2

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.

- I would like to provide some of my observations about how to improve the clusterKptMatchesWithROI function developed for the project in Section 4. While this method enhances the input data quality for TTC calculations, further improvements can be made by incorporating dynamic threshold adjustments based on the vehicle's speed or environmental conditions. For now, I hope the current implementation satisfies the project requirements, but I wanted to share my thoughts for future improvements.
- The current provided matching bounding boxes process which is shared in FinalProject_Camera.cpp starting from line 268 might have presented some inefficient approach. Using hash maps for direct access based on boxID in C++ significantly enhances efficiency in searching and matching operations, primarily due to the constant time lookup it offers, as opposed to the linear time complexity of traditional search methods like the current one that provides a matching process. Since a solution has been provided I did not go further for this improvement.

- For the Shi-Thomasi detector, due to OpenCV (4.1.0) issue it could not work with the SIFT descriptor. Due to that reason, its corresponding section could not provided.
- Except for the AKAZE detector, all other detectors failed to work with AKAZE and SIFT descriptors.
- As it can be seen in Harris Figure 2, there are a couple of inf, NaN, and high computations. In addition to Harris, ORB Table 6 presents some problematic results. I think these are undesired results and occurred due to the result of the keypoint detection and matching process. As it can be seen in Fig.3, the number of detected key points inside the car in the ego lane (i did not provide the bounding box for it in the image) are not so informative and could be the reason for the inefficient results.

Table 1: Lidar-based TTC Estimation Results (seconds)

Image	**Lidar**
1	12.51
2	12.61
3	14.09
4	16.68
5	15.74
6	12.78
7	11.98
8	13.12
9	13.02
10	11.17
11	12.8
12	8.95
13	9.96
14	9.59
15	8.52
16	9.51
17	9.61
18	8.39

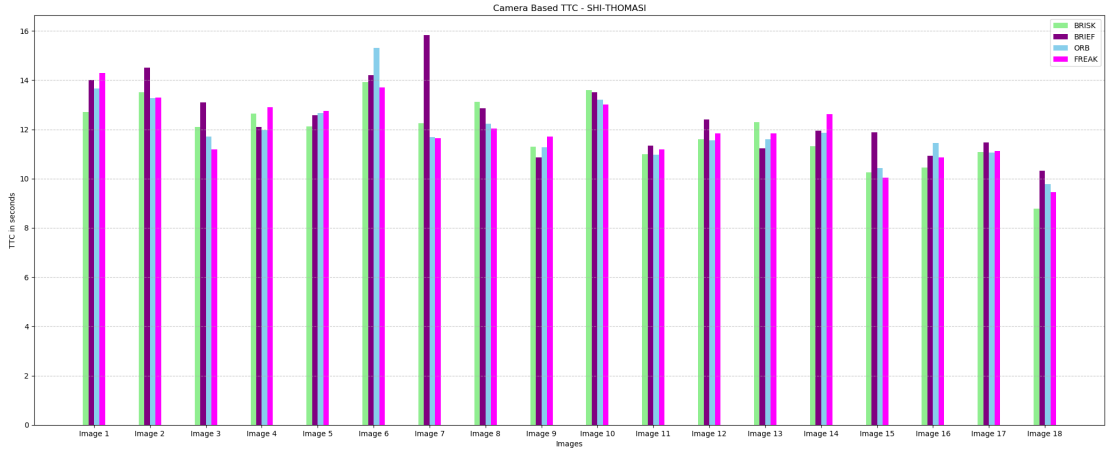

Figure 1: Camera based TTC durations - ShiThomasi

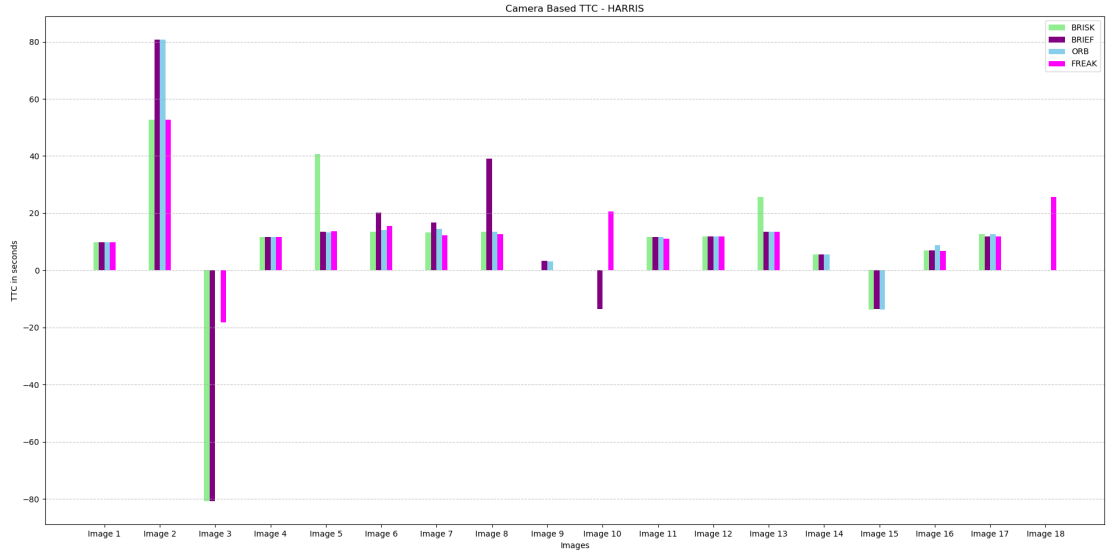

Figure 2: Camera based TTC durations - Harris

Figure 3: Harris Corner Detector Results

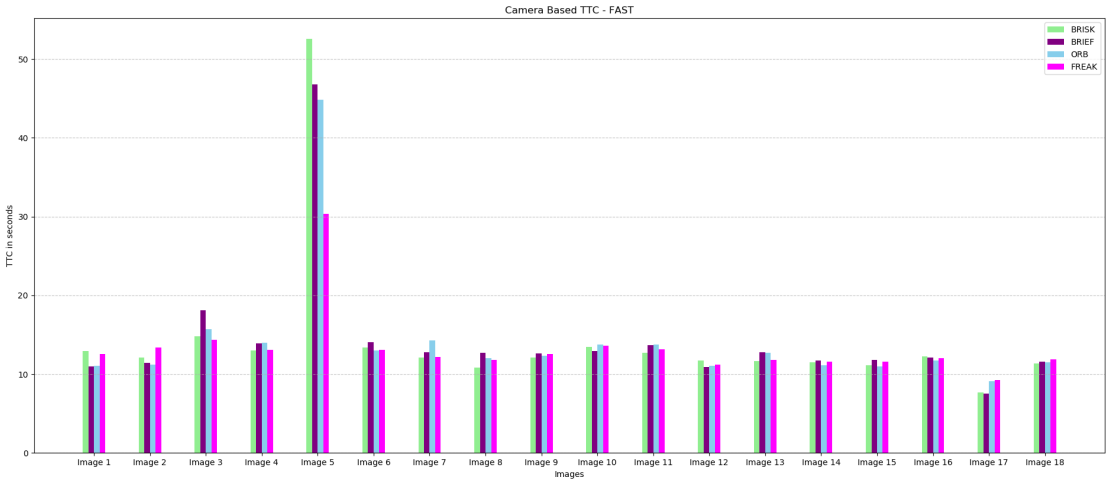

Figure 4: Camera based TTC durations - Fast

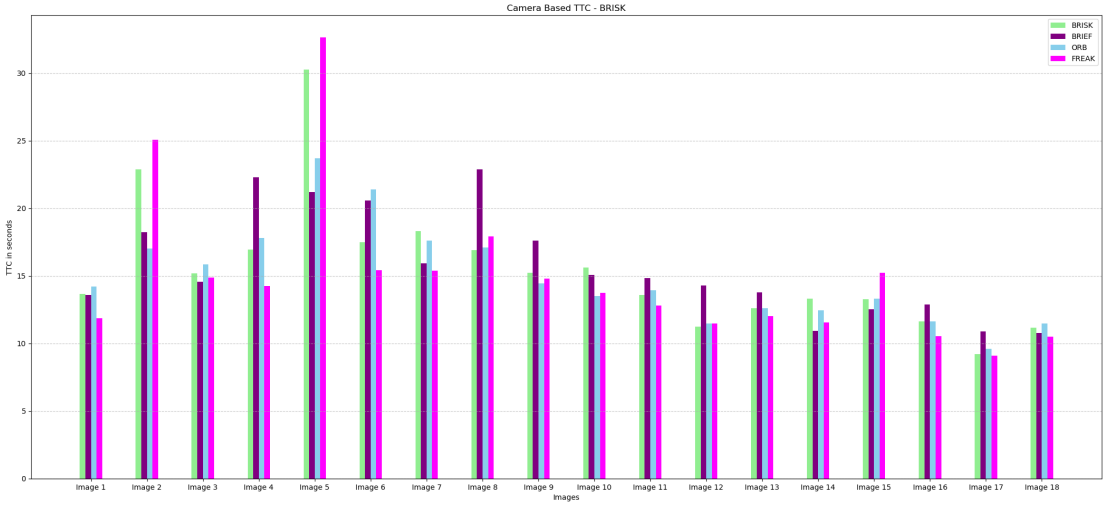

Figure 5: Camera based TTC durations - Brisk

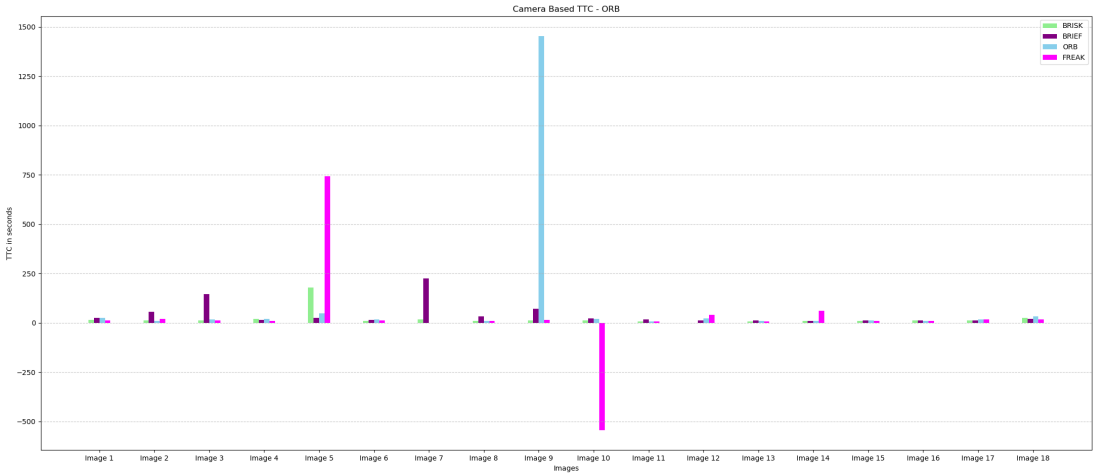

Figure 6: Camera based TTC durations - ORB

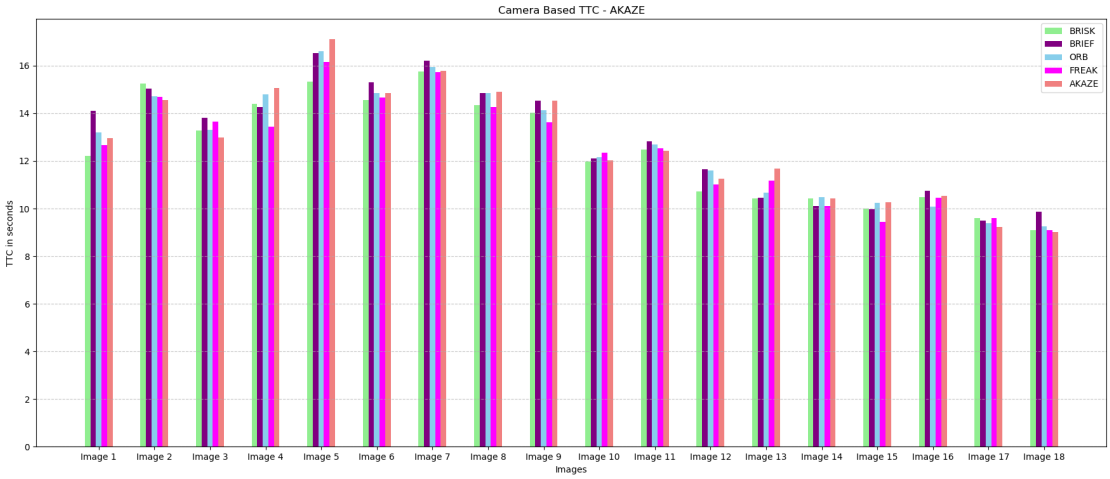

Figure 7: Camera based TTC durations - AKAZE

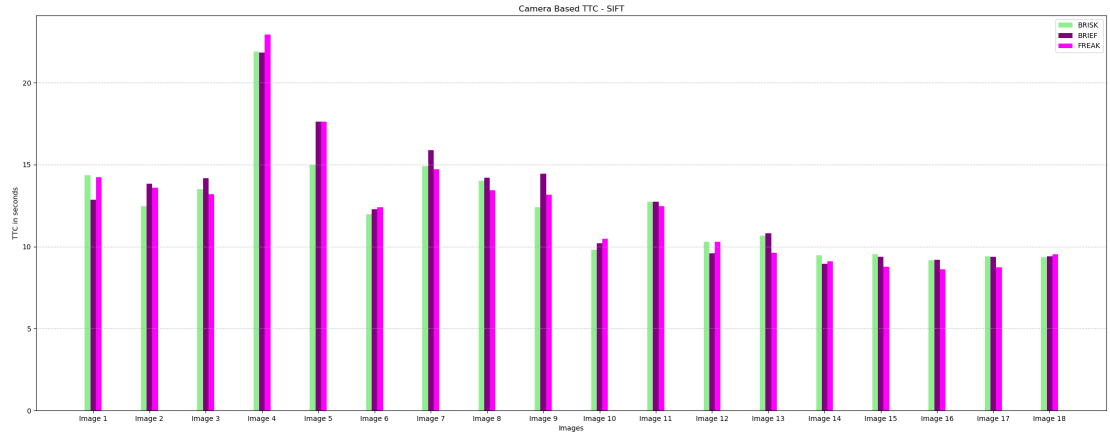

Figure 8: Camera based TTC durations - SIFT