Project Report – Estimation of Time to Collision (TTC)

In this project report I am discussing all the tasks that were provided as a part of Project 3 – Estimation of Time To Collision using Camera and Lidar Data take from the KITTI dataset.

Task 1: 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 in the camFusion_Student.cpp file. In this function we have to loop through all the matches between the prevoius and current image frames that were identified by keypoint mathcing algorithm. While looping through the matches, a keypoint is taken in the currect frame checked if it is present in any of the bounding boxes provide by OpenCV DL algorithm. If the keypoint is present in the bounding boxes both in the previous and the current frames then the correspoing boxId's were stored in a struct which holds the boxId from the previous frame and the current frame and the number of matches. Once looped trough all the matchs, the strcut that was previously created was refined to only provide only the boxId with max number of matches.

Task 2: Compute the time-to-collision in seconds for all matched 3D objects using only Lidar measurements from the matched bounding boxes between current and previous frame.

The code to compute the Time to collision based on lidar points is implemented in the function " computeTTCLidar" in the camFustion_Student.cpp file. Lidar points in the previous and current timestep is provided to the function allong with the frame rate at which it was captured. The lidar points are then passed through a median filter to estimate the distance of the vehicle. Median filter is preferred here so that outliers will not affect the calculation.

Task 3: 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.

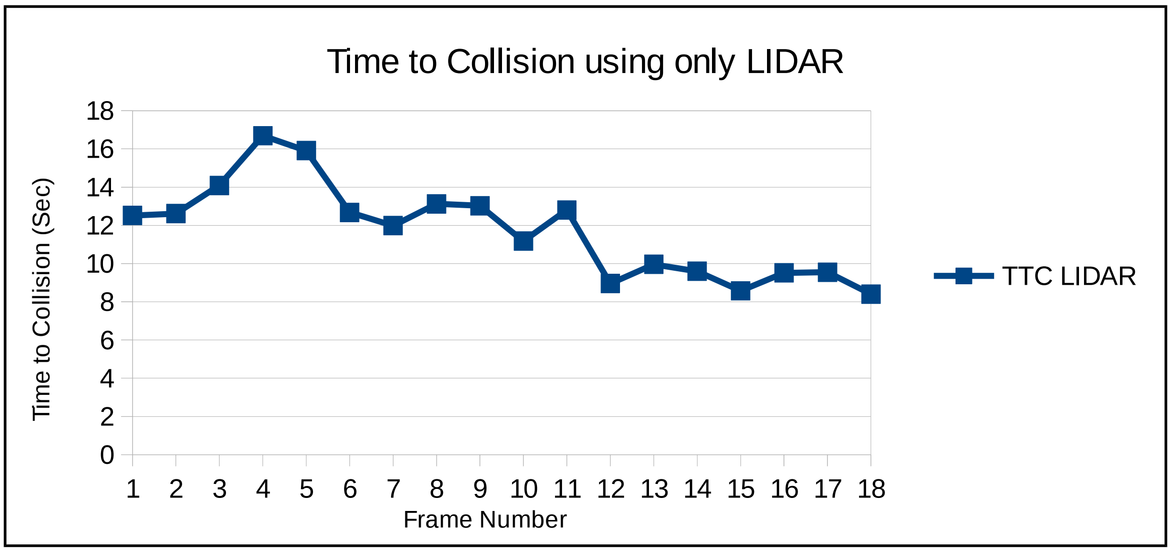
The method "clusterKptMatchesWithROI" was used to implement to cluster all the keypoints in a frame and associate the keypoints to particular bounding box in the same frame. To do this first a median distance of all the matches were calculate to eliminate the influence of outliers in the calculation. Once the median distance is calculated we loop through every keypoint matches in the previous and current frame and check if it is inside a give bounding box. If so, then it is added to a vector of keypoints within that bounding box which will be used later to calculate the TTC.

Task 4: 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.

The TTC from camera is calculated based on the keypoints that we have clustered in the previous step. We will loop though all the keypoint matches with the bounding box on the preceding vehicle and filter these keypoint matches using a median filter to avoid any erroneous matches that passed the previous filtering step. Once we filter all the values, we calculate the TTC using the median keypoint values.

Task 5: 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 the lidar data did not pass through much different filtering algorithm I am presenting a single graph which has the mean values of all the time steps of lidar data.



The above graph shows the TTC calculation based on lidar in different time steps. There are two particular time which draws my attention.

1) Frames 2, 3, 4 show an increasing time to collision which is really odd give that the ego car is getting close to the other in every time step.
2) In frame 5 & 6 and frames 11 & 12 we can see that there is a very steep decrease in TTC calculation of about 4 seconds which might either be a false positive or false negative.

The reason I feel why these abnormalities are popping up in our calculation are

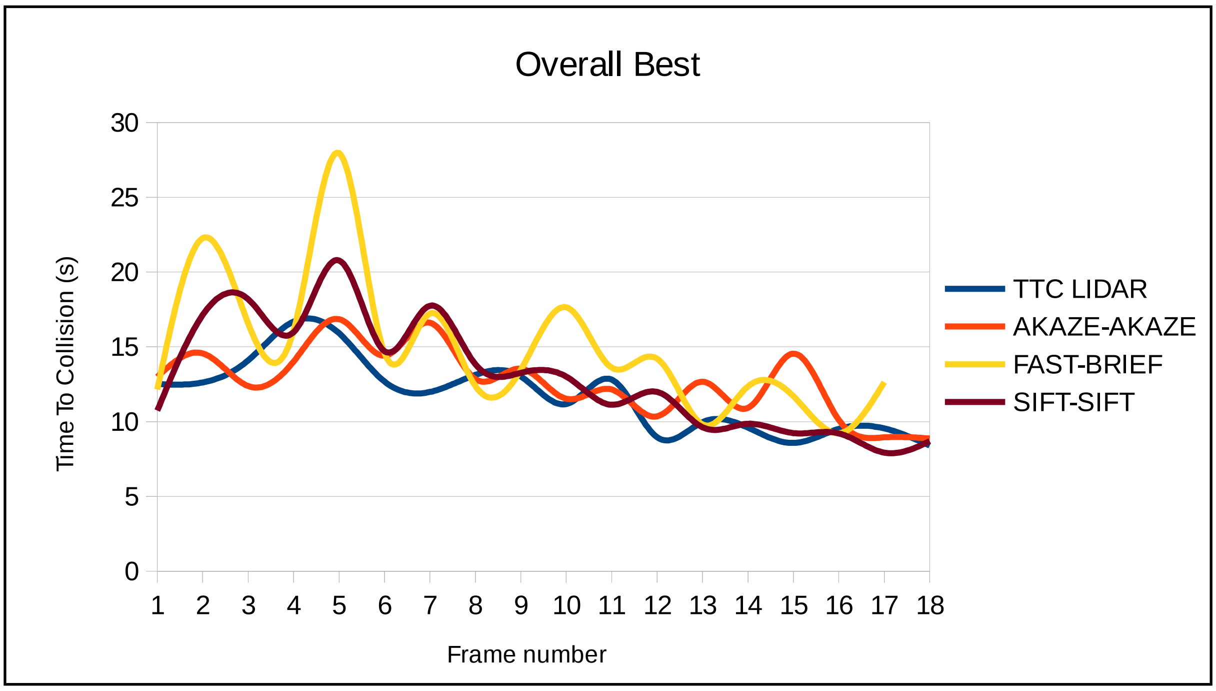
1. The TTC calculation assumes a constant velocity model which is not applicable to a real life scenario where a relative velocity and most important relative acceleration/deceleration difference are noticed. So we need to account in for the relative acceleration in our calculation.
2. The lidar data is not properly passed though a filter except for cropping the road surface using a cropbox. We need to filter the cropbox using a spatial neighborhood filter method which will make sure to remove all the spurious points from the lidar scan data.

Task 6: 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.

Different detector/descriptor combinations were tried and all the TTC values are tabulated and show in the comparison_repot.ods file in the same director. Since the table is too large I am not includeing it here.

Out of all the comparisons made, the best combination of detector/descriptor that I think will work best in the scenario are

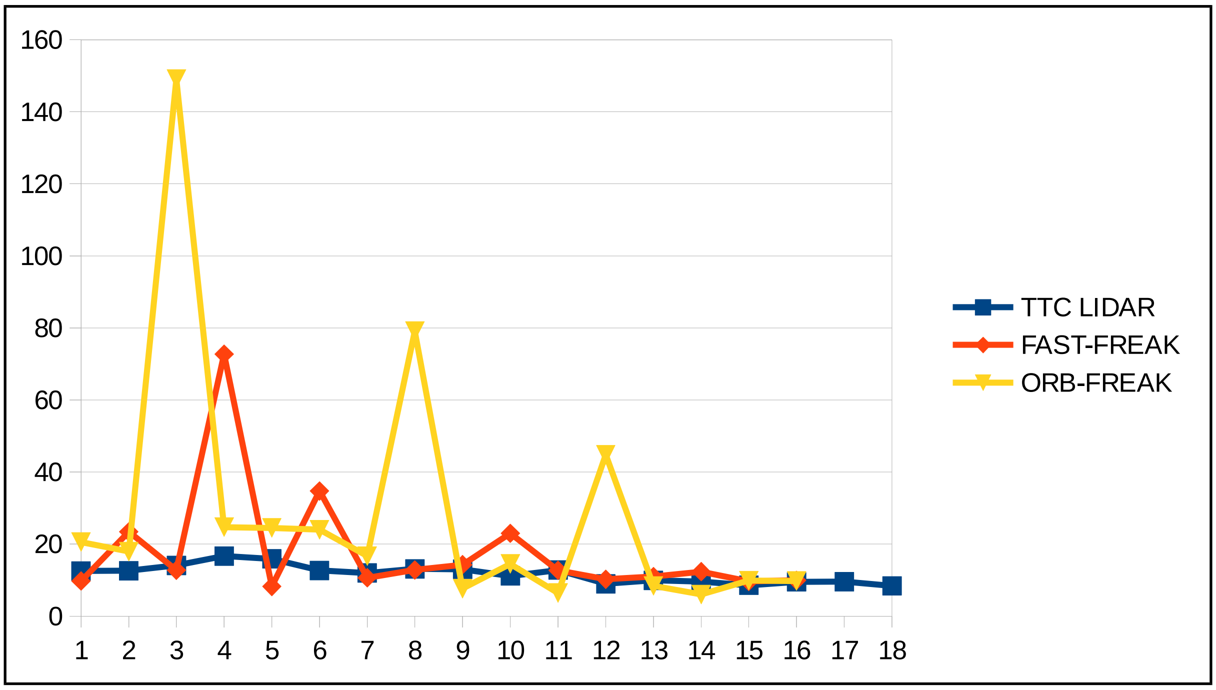
1. AKAZE – AKAZE
2. FAST – BRIEF
3. SIFT SIFT



Although you can see that there are a lot of spikes in the TTC values calculated, the above mentioned detector-descriptor combinations are the ones with the least spike and aligns closely with the TTC calculated by LIDAR only method. Here I take TTC lidar to be my base for comparision since I think the lidar based calculations are far more superior compared to the camera based method.

<u>Example of where the camera TTC was way off from the mean</u>

You can see from the graph below where there are two instance where a detector-descriptor pair was way off from the TTC calculated by LIDAR. From this I can say that for every use case we need to find a best detector-descriptor pair which will work for that particular use case.


TTC LIDAR
FAST-FREAK
ORB-FREAK