Track an Object in 3D Space: Time to Collision Estimation

Seymur Dadashov

July 21, 2020

Abstract

The purpose of the project is to combine different sensors (i.e. LiDAR, camera) on a vehicle to evaluate the Time-to-Collision (TTC) value for a Collision-Avoidance-System (CAS). The TTC is estimated using two different approaches that involves the use of only a LiDAR and only a camera. The two methods are compared and contrasted as well as the detectors and descriptors used to generate keypoints for camera based estimation. The sensors are fused to together through the use of the deep learning YOLO (You Only Look Once) algorithms to identify vehicles in the camera image and form bounding boxes around those images. The bounding boxes serve as a key step in order to efficiently filter LiDAR points and keypoints that pertain only to the preceding vehicle that is used for the calculation of the TTC.

1 Task FP.1 Match 3D Objects

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 tempoary vector, and store the best points in the `bbMatches` map.

2 Task FP.2 Compute LiDAR-based TTC

In order to estimate the Time-to-Collision value using the LiDAR, one must compute the measurements between LiDAR points in the previous and current time step. The time step allows for a displacement value to be measured between lidar points and therefore an estimate can be calculated for TTC. The following equations describe the observed behaviour of the vehicle between time steps and the approach to calculating TTC:

$$d(t + \Delta t) = d(t) - V_0 \cdot \Delta t \tag{1a}$$

$$V_0 = \frac{d(t) - d(t + \Delta t)}{\Delta t} = \frac{d_0 - d_1}{\Delta t} \tag{1b}$$

$$TTC = \frac{d_1}{V_0} = \frac{d_1 \cdot \Delta t}{d_0 - d_1} \tag{1c}$$

The equations above consider a single point from each time step for the calculation; however, the LiDAR provides multiple points in space that measure the distance to the target object. Typically in a situation like this, one can either take the mean or median of all the points that lie along the x-direction of the LiDAR point cloud and that correspond to the rear tail gate of the preceding vehicle.

The code implementation was done in `camFusion_Student.cpp` on line 235 and uses the median approach as most of the data is structured into a cluster in space along some median x value that corresponds to the rear of the preceding vehicle. The mean in this case is not a sufficient solution as it cannot handle outliers that may occur from the LiDAR picking up road measurements farther in front of the vehicle and still remain within its bounding box since we are dealing with 3D space. As seen in the code implementation in List. 1, the absolute value is compared between the median value in the previous frame and the current frame in order to see if the preceding vehicle is in fact slowing down at a constant velocity or speeding back up. The comparison to *double epsilon* scaled by the maximum value between the two medians is done in order to see if the their absolute difference is approximately 0. If so the difference is too small, TTC is assigned NAN to indicate that the preceding vehicle is at a stand still. Similarly, if the difference is larger than epsilon, but still close to 0, there will be large spikes in positive and negative values of TTC. This was observed experimentally, thus another check is added to restrict TTC values to be in the range $TTC <= \pm 20s$, and values that are considered spikes due to the inverse relationship scene in Eqn. (1c) will be assigned NAN as well.

```cpp
/void computeTTCLidar(std::vector<LidarPoint> &lidarPointsPrev,
                     std::vector<LidarPoint> &lidarPointsCurr, double frameRate, double &TTC)
{
    double dT = 1/frameRate;
    double medianXPrev = 0.0;
    double medianXCurr = 0.0;
    std::vector<double> sortXPrev;
    std::vector<double> sortXCurr;

    for(auto point : lidarPointsPrev) {sortXPrev.push_back(point.x);}
    for(auto point : lidarPointsCurr) {sortXCurr.push_back(point.x);}

    sort(sortXPrev.begin(), sortXPrev.end());
    sort(sortXCurr.begin(), sortXCurr.end());

    medianXPrev = (sortXPrev.size() % 2 == 0) ? (sortXPrev[sortXPrev.size()/2 - 1] +
                sortXPrev[sortXPrev.size()/2]) / 2 : sortXPrev[sortXPrev.size()/2];
    medianXCurr = (sortXCurr.size() % 2 == 0) ? (sortXCurr[sortXCurr.size()/2 - 1] +
                sortXCurr[sortXCurr.size()/2]) / 2 : sortXCurr[sortXCurr.size()/2];

    if(fabs(medianXPrev - medianXCurr) <= max(medianXPrev,medianXCurr)*__DBL_EPSILON__){
        TTC = NAN;
    }
    else{
        TTC = medianXCurr*dT / (medianXPrev - medianXCurr);
        if(TTC <= -20 || TTC >= 20){TTC = NAN;}
    }
}
```

Listing 1: Calculation of TTC using LiDAR implementation in C++

The function `cropLidarPoints` in `lidarData.cpp` that initially filtered for the ego lane did not take into account the LiDAR taking measurements far in-front of the vehicle, despite being within the bounds of the top down view $0 - 20m$ range in the x-direction as seen in Fig. 1. To solve this issue, the median was used with a boundary parameter to filter points too far outside the range of the median as this problem occurred downstream when the preceding vehicle came very close to car. Figure 2 shows the ideal behaviour of how LiDAR points should be cropped.

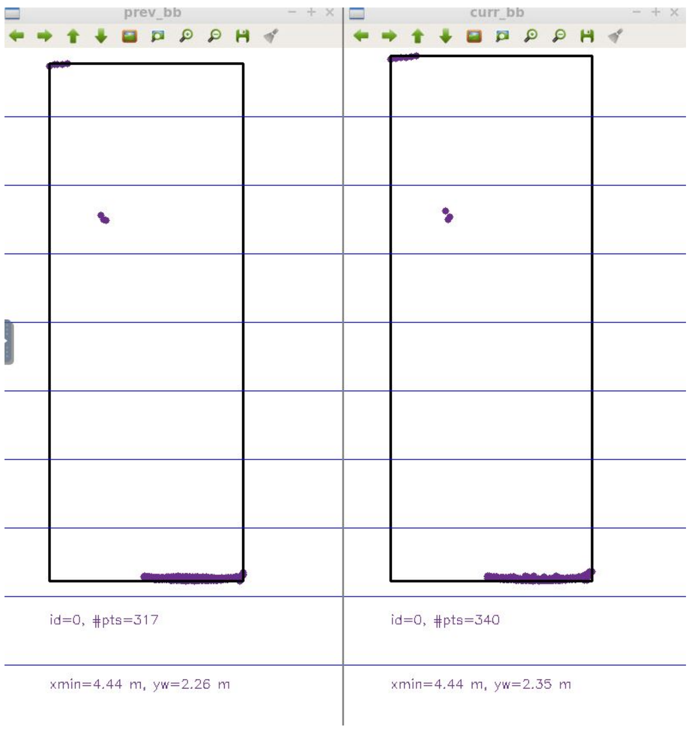


Figure 1: Outliers caused by LiDAR detection of the road.

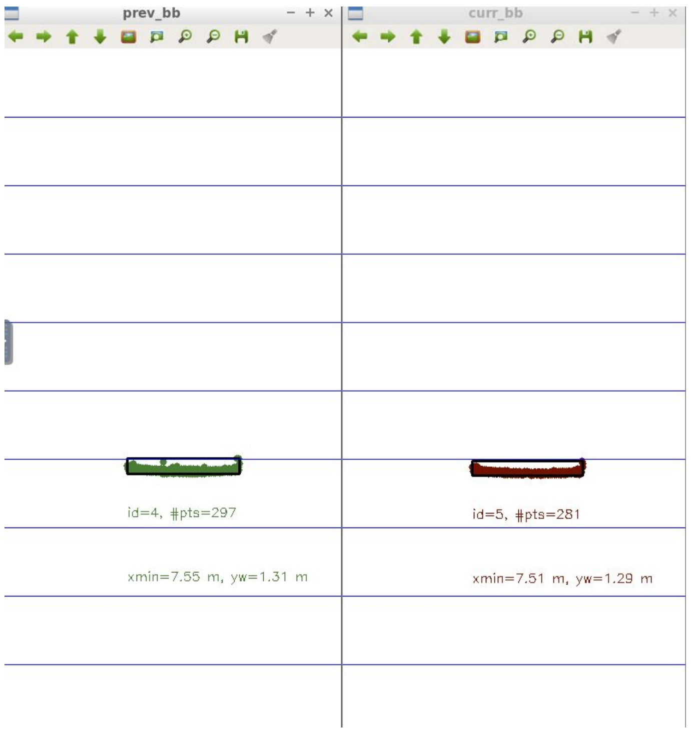


Figure 2: Lidar points correctly bounded and outliers removed

3 Task FP.3 Associate Keypoint Correspondences with Bounding Boxes

In order to associate keypoints correspondences with bounding boxes, the keypoints for a given match need to be contained within the same bounding box in both current and previous frames. The procedure is a simple evaluation using the `contains()` method; however, this solution would not account for potential outliers i.e. the False-Positives that exist within the matched pairs.

To filter out the False-Positives, we first evaluate the average distance each keypoint has shifted between the previous frame and the current frame and scale it by a certain percentage to provide a boundary around the mean value. The norm between a pair of keypoints is evaluated to see if it belongs within the mean scaled distance and only values within this range are considered correspondences. The False-Positives will have a very large norm compared to the rest due to the mismatch and thus their position in the 2D pixel space will be significantly farther apart. The function `clusterKptMatchesWithROI` completes this task on line 135 of `camFusion_Student.cpp`.

4 Task FP.4 Compute Camera-based TTC

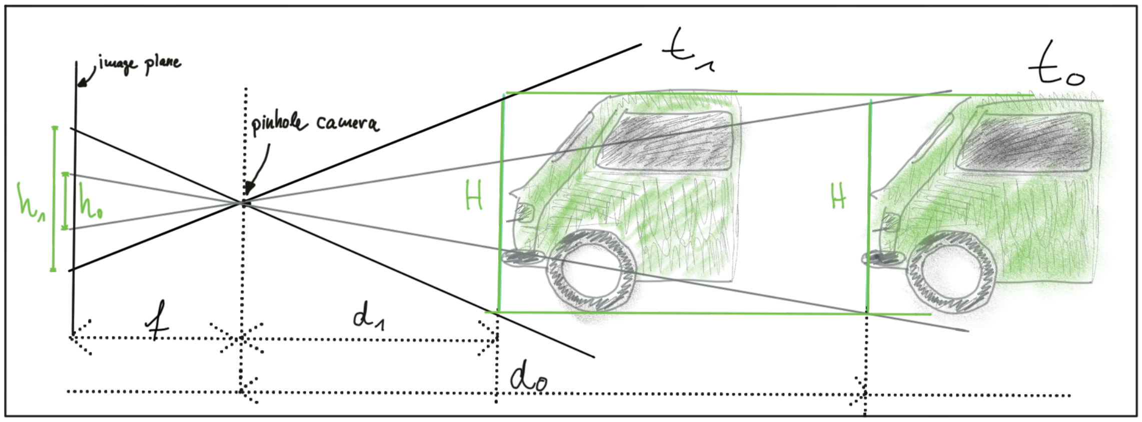


Figure 3: Pinhole Camera Diagram for Projecting Object Into Image Plane

In order to understand how to compute the TTC using a camera based approach, the relationship between the motion of the object and the image of the object need to be established. `Structure-from-Motion` is the way that one can extract a motion model from an image so long as the object in the image always stay in motion. The motion itself can be related by the following equation and relates the displacement of the object to the size of the object in the image plane, hence the ratio of $\frac{h_1}{h_0}$.

Project the object into the camera:

$$h_0 = \frac{f \cdot H}{d_0}, \quad h_1 = \frac{f \cdot H}{d_1} \tag{2a}$$

Relate projection and distance:

$$\frac{h_1}{h_0} = \frac{\frac{f \cdot H}{d_1}}{\frac{f \cdot H}{d_0}} = \frac{d_0}{d_1} \quad \rightarrow \quad d_0 = d_1 \cdot \frac{h_1}{h_0} \tag{2b}$$

Substitute in constant-velocity model:

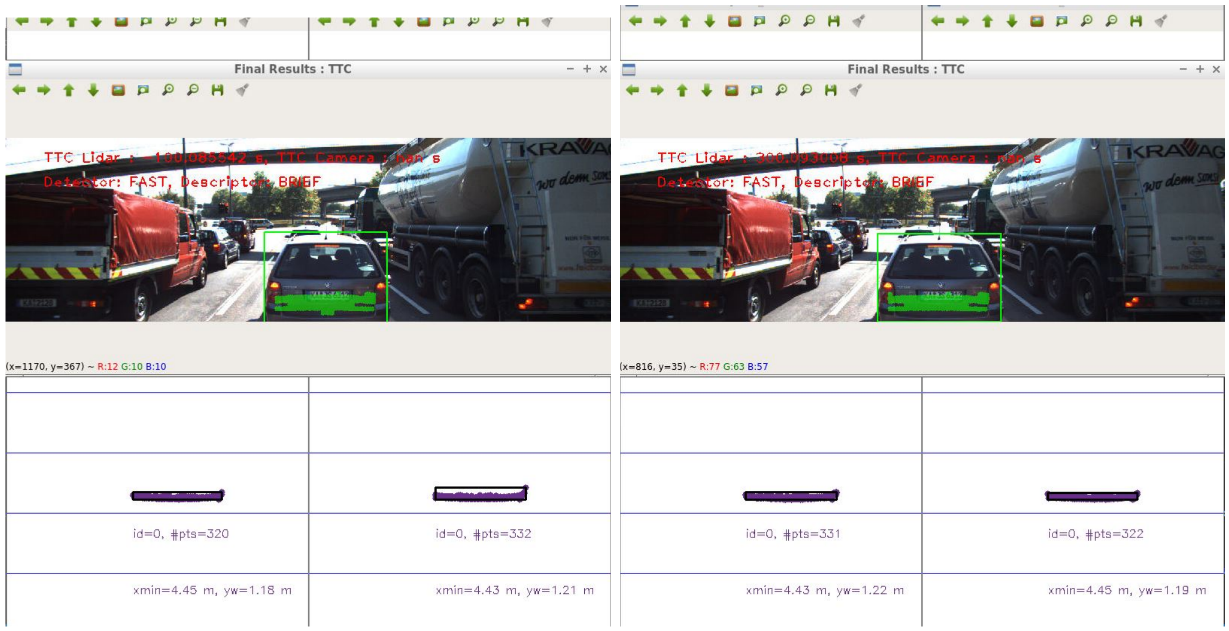
$$d_1 = d_0 - V_0 \cdot \Delta t = d_1 \cdot \frac{h_1}{h_0} - V_0 \cdot \Delta t \quad \rightarrow \quad d_1 = \frac{-V_0 \cdot \Delta t}{(1 - \frac{h_1}{h_0})} \tag{2c}$$

Compute the Time-to-Collision:

$$TTC = \frac{d_1}{V_0} = \frac{-\Delta t}{(1 - \frac{h_1}{h_0})} \tag{2d}$$

The implementation is achieved by measuring the ratio of how the distance between keypoints changes from the previous frame to the current frame. If the keypoint ratio increases as we increase time-steps, then we can assume the object is getting larger in the image plane and closer in 3D space. The rate at which the ratio increases or decreases by depicts how quickly the object is approaching. However, the estimation is only as good as the motion that the camera is seeing. In other words, the preceding vehicle needs to be moving at a different velocity than the ego vehicle in order to properly determine a TTC value. If their is little observed change through the camera, i.e. the ratio $\frac{h_1}{h_0}$ stay relatively close, then the denominator approaches *infinite*. Experimentally, this is what was observed in the image series provided by KITTI since in slow moving traffic the camera does not see a rapid change in the size of the preceding vehicle in the image plane. Thus, the estimated TTC is always higher or at times infinite when compared to what the LiDAR is reading.

5 Task FP.5 Performance Evaluation 1



```
TTC Lidar : -100.085542 s, TTC Camera : nan s
Detector: FAST, Descriptor: BRIEF
(x=1170, y=367) ~ R:12 G:10 B:10
id=0, #pts=320    xmin=4.45 m, yw=1.18 m
id=0, #pts=332    xmin=4.43 m, yw=1.21 m

TTC Lidar : 300.093008 s, TTC Camera : nan s
Detector: FAST, Descriptor: BRIEF
(x=816, y=35) ~ R:77 G:63 B:57
id=0, #pts=331    xmin=4.43 m, yw=1.22 m
id=0, #pts=322    xmin=4.45 m, yw=1.19 m
```

Figure 4: LiDAR Calculations over Small Distances

The scenarios where a TTC estimate is unable to be properly calculated occurs when the preceding vehicle is standing still or barely moving. The result for this is a direct result of how we modeled the motion equations and thus the calculation of the TTC. The LiDAR relies heavily on the difference between LiDAR measurements between successive frames and if the measurements return very similar values between a single time-step, the resulting Eqn. 1c approaches *infinite* as the denominator goes to 0. However, unlike the camera implementation where we see infinite value register from the calculations, the LiDAR tends to oscillate very close to 0, but rarely goes to finite value of 0. Therefore, the numerator experiences large magnifications in value returning TTCs that make little to no sense as seen in Fig. 4. As stated in task FP.2, a check is used to determine if the LiDAR is oscillating close to 0 or has spiked to a large value and set the TTC to NAN. The preceding vehicle is then considered to be not moving if both TTC based on LiDAR and camera return similar results.

6 Task FP.6 Performance Evaluation 2

The combination of all detectors and descriptors can be found in `Final_Project_Data_Log_Save.csv` and further processed data can be found in `Final_Project_Data_Processed.xlsx`. Analyzing the results from multiple combinations, it can be seen that detectors that discover a larger number of keypoints and thus resulting in a higher number of matches produce results that can be properly processed for a potentially meaningful TTC estimate. Specifically, two chosen detectors can be seen with all the corresponding descriptors pairs with them in Figures 5 & 6. Figure 5 displays FAST and it can be clearly seen that there is a downward trend as we move through the frames, which is expected as the vehicles slows down to a near stop by the end. The data had several large outliers removed, but overall can perhaps be used for a rough estimate of the TTC. On the other hand, Figure 6 has no downward trend and is actually oscillating if all the data from all ranges was included. The reason for the HARRIS detector struggling to estimate the distance frame by frame is due to the extremely small number of keypoints that is actually discovered in the image. FAST generates almost 10 times and it can be seen how having a small number of points to base measurements off of would result in poor accuracy in timing estimation.

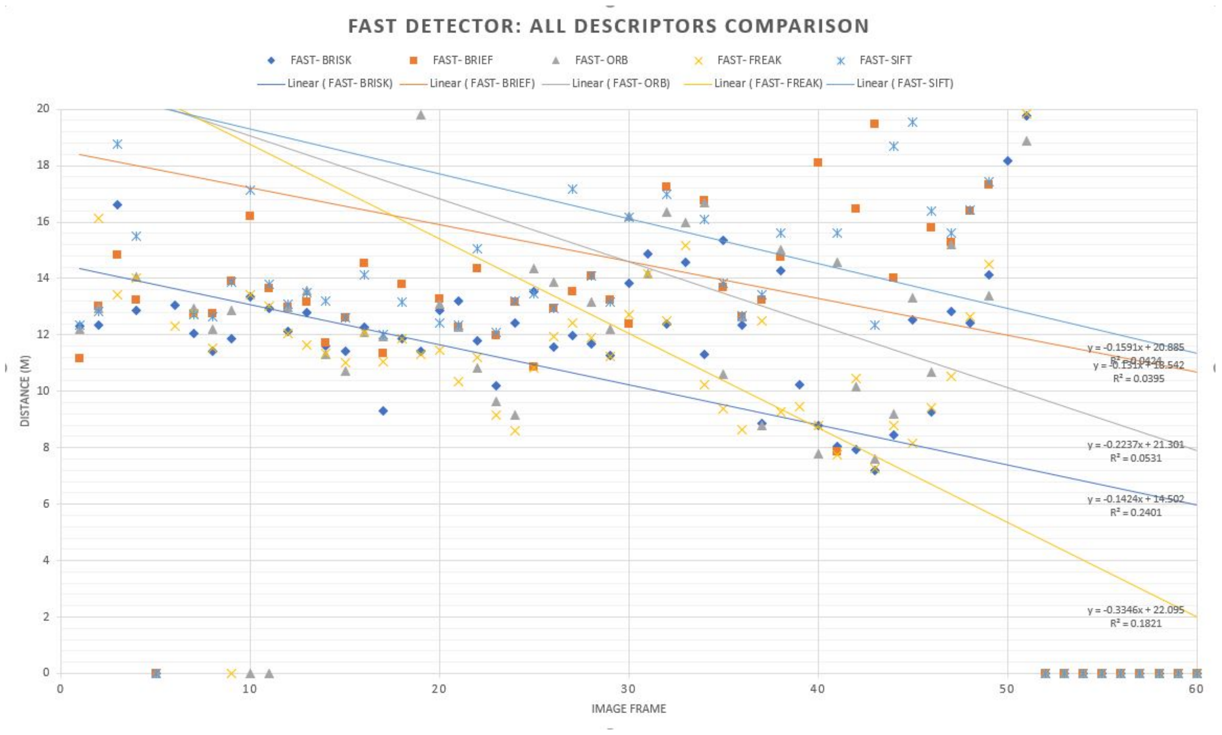


Figure 5: Plot of distance based on keypoints detected by FAST and corresponding descriptor combinations.

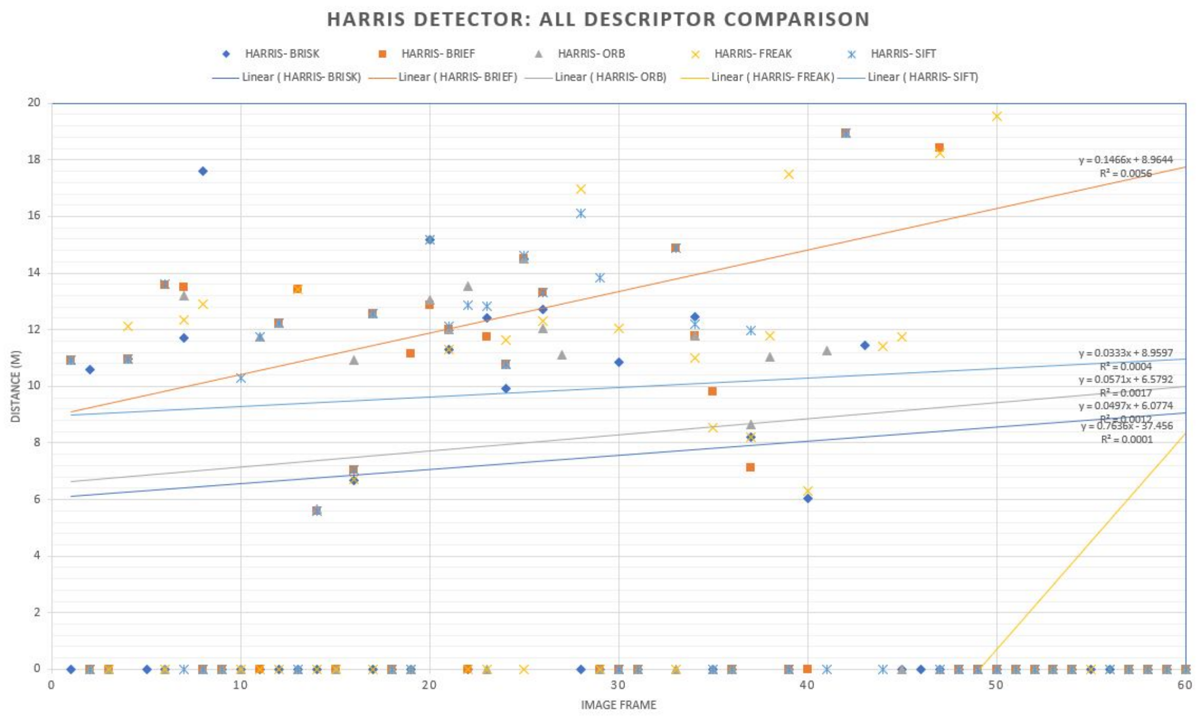


Figure 6: Plot of distance based on keypoints detected by HARRIS and corresponding descriptor combinations.