3D Object Tracking - Final-Term Project

The goal of this project is to track 3D object bounding boxes over time frames by looking at the keypoint correspondences within those boxes. Once the tracking is achieved, then the **Time to Collision (TTC)** for the preceding vehicle is computed both by using LIDAR and camera sensor data. The difference between the two measurements is noted for various combinations of keypoint detectors and descriptors which are used to estimate the camera based TTC. The following **rubric points** are addressed in the project.

FP.1| Match 3D Objects

"**matchBoundingBoxes()**" function has been implemented in the file camfusion_Student.cpp which takes input both previous and cureent data frames and outputs ids of the matched ROI. We use a **multimap** to store the bounding box's ID of both current and previous frames which contains the matched pair. For each current bounding box, we find the corresponding previous bounding box which contains the maximum number of matched pair. We use `equal_range()` function in multimap to find all the pairs having the same current bounding box ID. The output is a **map** of previous bbox ID which is having the maximum number of matched pair to each current bbox.

```
 for (int cid : currFrameBboxIds) {
    auto prevMatchedIds = bboxIdMap.equal_range(cid);
    std::vector<int> countMaxId(prevMaxId + 1, 0);
    for (auto it = prevMatchedIds.first; it != prevMatchedIds.second; it++) {
        if (it->second != -1) {
            countMaxId[it->second] += 1;
        }
    }
    int pid = std::distance(countMaxId.begin(), std::max_element(countMaxId.begin(),
countMaxId.end()));
    bbBestMatches.insert(std::make_pair(pid, cid));
}
```

FP.2| Compute Lidar-based TTC

The Time-to-Collision for all the matched 3D objects have been computed in seconds using Lidar measurements from matched bounding boxes between current and previous frame using the function **computeTTCLidar()** in the file camfusion_Student.cpp. The formula to compute TTC is `TTC = dT*d1/(d2 - d1)` where **d1** is the distance to the preceding vehicle in the current frame and **d2** is the distance in the previous frame. In order to get away with outliers, we compute the TTC from the *median* of all the Lidar points, which then gives a quite stable estimate of TTC.

FP.3| Associate Keypoint correspondences with Bounding Boxes

In order to associate the keypoint correspondences with the bounding box we loop across all the matched keypoints and store the keypoints of the current frame which are enclosed inside the given bounding box.

```
double sumDist = 0.0;
for (auto &match : kptMatches) {
    if (boundingBox.roi.contains(kptsCurr[match.trainIdx].pt)) {
```

```
        boundingBox.kptMatches.emplace_back(match);
        sumDist += match.distance;
    }
}
```

In order to eliminate the outliers among the matches, we compute a robust mean of all the euclidean distances between the corresponding matches and the points which are too far away from the mean are removed. Using `match.distance`, we compute the euclidean distance.

FP.4| Compute Camera-based TTC

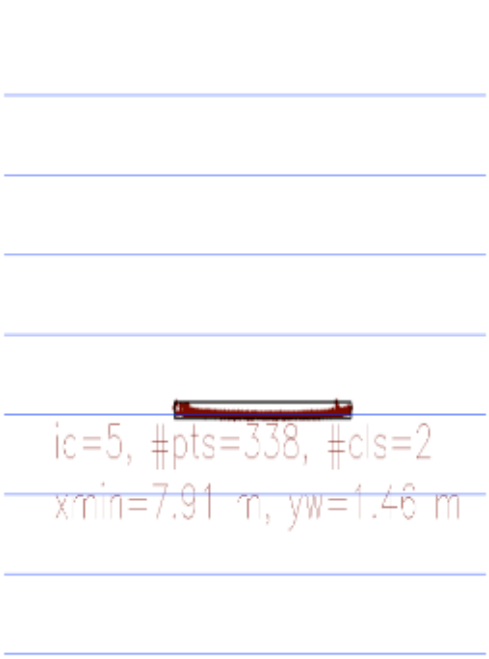
The Time-to-Collision for all the matched 3D objects have been computed using only **keypoint correspondences** from the matched bounding boxes between current and previous frames. The formula used to compute the camera-based TTC is `TTC = -dT/(1 - distRatio)` where, *dT = 1.0/frameRate* and distRatios is the ratio of the relative distance between the keypoints in the current and previous frames. In ordeer to avoid the intruding of outliers in the calculation, we consider only the median of the keypoints.
This is the following implementation to compute the TTC.

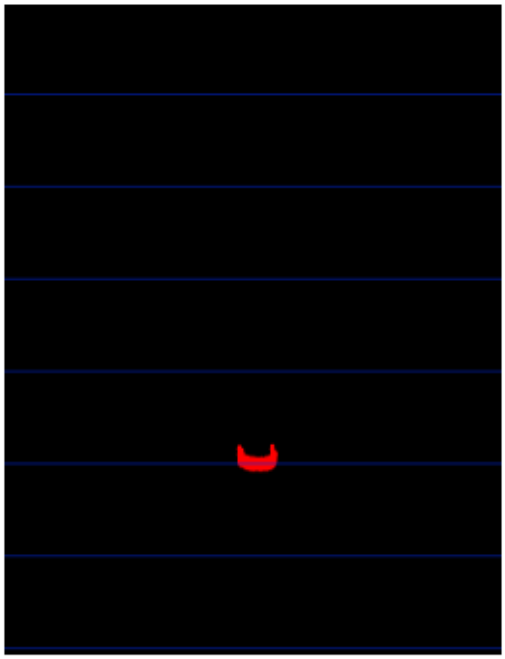
```
for (auto it1 = kptMatches.begin(); it1 != kptMatches.end()-1; ++it1) {
    cv::KeyPoint keyCurrOuter = kptsCurr[it1->trainIdx];
    cv::KeyPoint keyPrevOuter = kptsPrev[it1->queryIdx];
    for (auto it2 = kptMatches.begin() + 1; it2 != kptMatches.end(); ++it2) {
        double minDist = 100.0;
        cv::KeyPoint keyCurrInner = kptsCurr[it2->trainIdx];
        cv::KeyPoint keyPrevInner = kptsPrev[it2->queryIdx];
        double distCurr = cv::norm(keyCurrOuter.pt - keyCurrInner.pt);
        double distPrev = cv::norm(keyPrevOuter.pt - keyPrevInner.pt)
        if (distPrev > std::numeric_limits<double>::epsilon() && distCurr >= minDist)
{
            double distRatio = distCurr/distPrev;
            distRatios.push_back(distRatio);
        }
    }
}
```

The descriptors BRISK, BRIEF, ORB, FREAK and SIFT are implemented in the function **descKeypoints** inside the file **matching2D_student.cpp**.

FP.5| Performance Evaluation 1

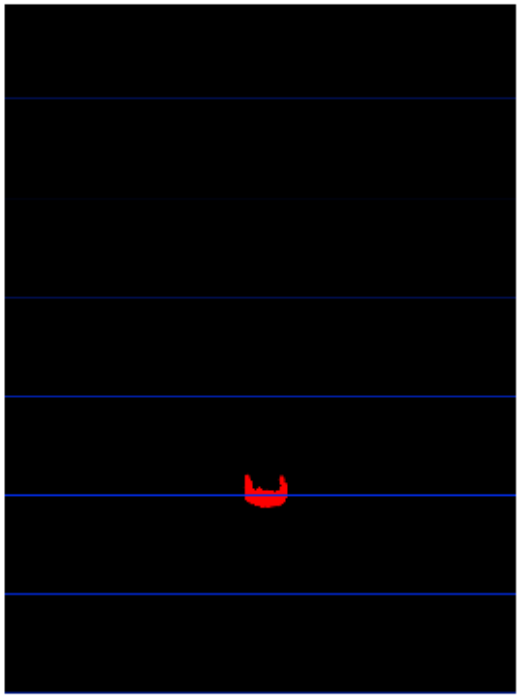
The TTC estimate of the Lidar sensor depends on the distance to the lidar point on the preceding vehicle. Since we have considered the **median** of all the points, we have ensured that outliers don't disturb the TTC computation, thereby we observe a stable estimate. The range of the TTC computed is from about 8 to 16 seconds. The results are shown in the spreadsheet **ResultsTTC.csv**. Below shows few examples of top-view perspective of Lidar points. **1)**


ic=5, #pts=338, #cls=2
xmin=7.91 m, yw=1.46 m

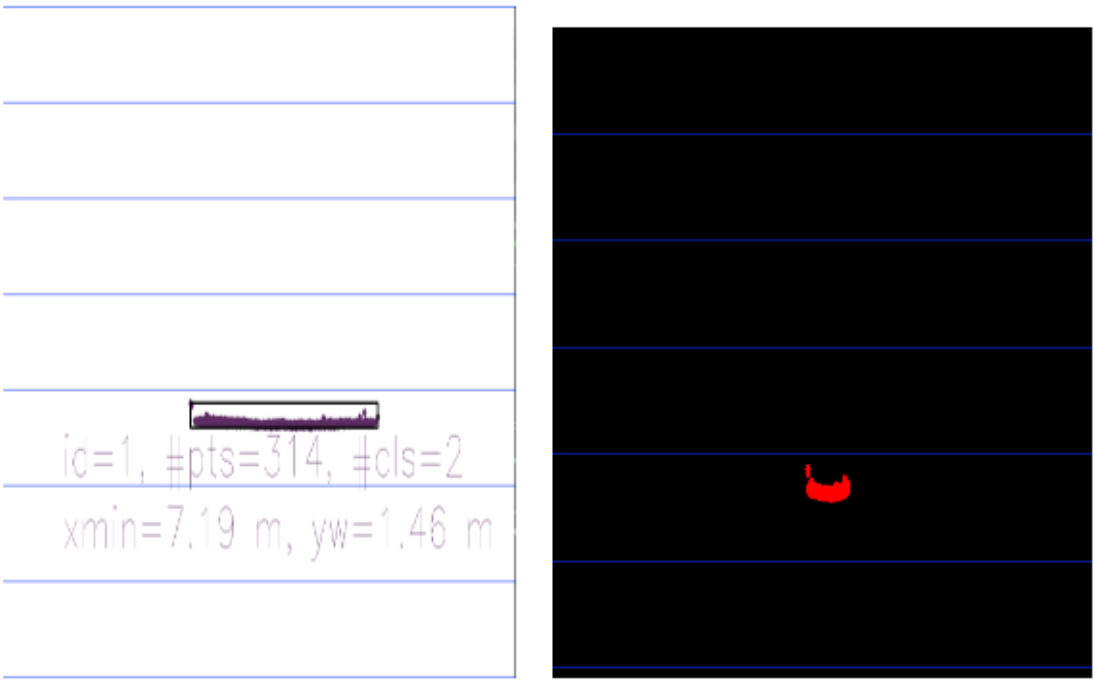


2)

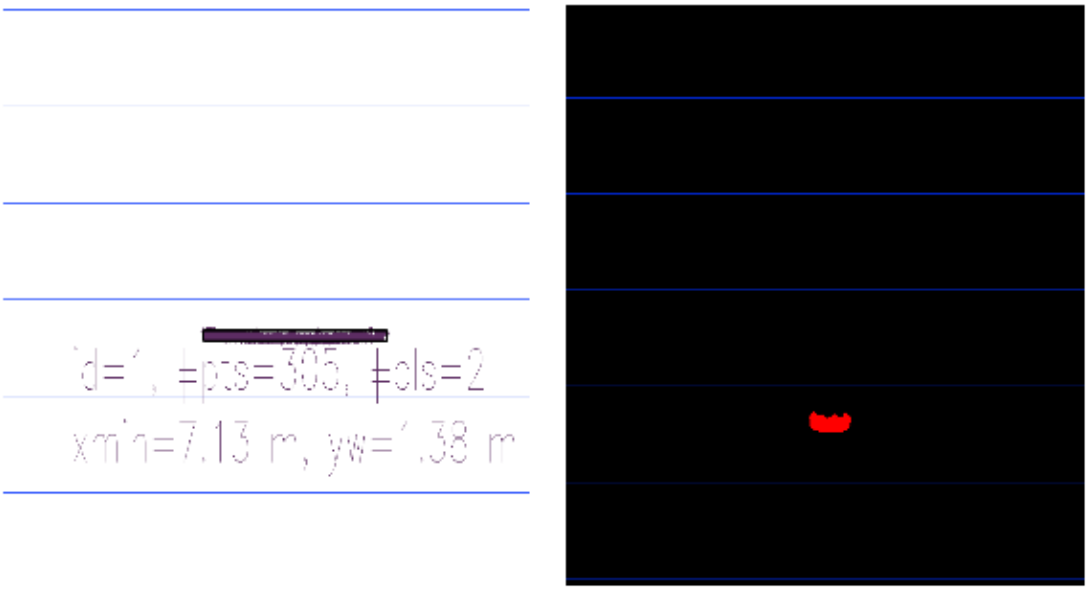

id=3, #pts=305, #cls=2
xmin=7.85 m, yw=1.46 m



3)



4)



FP.6| Performance Evaluation 2

The camera-based TTC has been computed for all combinations of detectors and descriptors we have implemented to find out in which case it performs the best and gives a stable TTC estimate. The comparsion results are shown in **ResultsTTC.csv**, which compares the TTC difference (**TTC_Diff**) for Lidar and camera. `TTC = TTC_Lidar - TTC_camera` The detector/descriptor pair for which the TTC_Diff is the smallest and

does not go way off is the best one. **Harris** and **ORB** detectors performs the worst as they result in unstable TTC estimate which are way-off. **AKAZE** and **SIFT** detectors performs the best and gives thd most stable TTC estimate. The results produced by the **SIFT detector** and **AKAZE descriptor** is shown below.

frameIndex	detector Type	descriptor Type	TTC_Lidar(in sec)	TTC_Camera(in sec)	TTC_Diff
1	SIFT	AKAZE	12.5156	11.542	0.973558
2	SIFT	AKAZE	12.6142	13.5986	-0.984366
3	SIFT	AKAZE	14.091	12.3504	1.74065
4	SIFT	AKAZE	16.6894	16.1913	0.498037
5	SIFT	AKAZE	15.7465	16.1427	-0.396111
6	SIFT	AKAZE	12.7835	11.2992	1.4843
7	SIFT	AKAZE	11.9844	14.2273	-2.24294
8	SIFT	AKAZE	13.1241	15.3854	-2.26124
9	SIFT	AKAZE	13.0241	12.7337	0.290439
10	SIFT	AKAZE	11.1746	11.3858	-0.211207
11	SIFT	AKAZE	12.8086	12.3477	0.460898
12	SIFT	AKAZE	8.95978	12.0373	-3.07753
13	SIFT	AKAZE	9.96439	9.98467	-0.02775
14	SIFT	AKAZE	9.59863	10.4011	-0.80243
15	SIFT	AKAZE	8.52157	9.26573	-0.744166
16	SIFT	AKAZE	9.51552	10.5265	-1.01102
17	SIFT	AKAZE	9.61241	8.276061	1.33636
18	SIFT	AKAZE	8.3988	9.2414	-0.842596