3D OBJECT TRACKING
SENSOR FUSION PROJECT #3

Jasmine L. Taketa-Tran

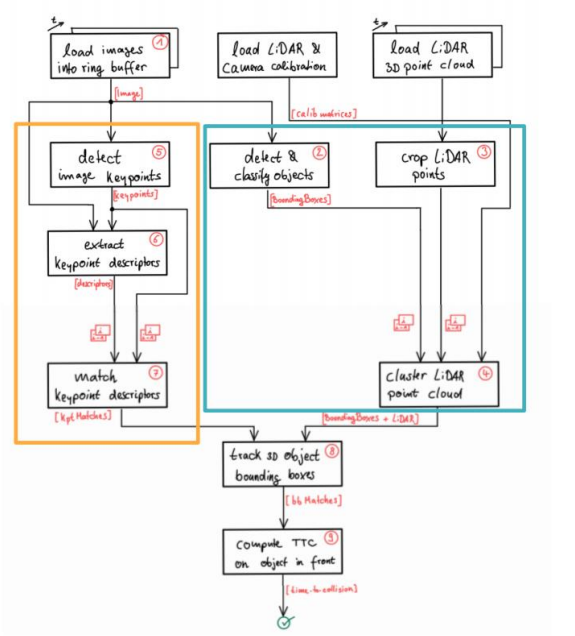
OVERVIEW

The objective of this project is to process 2-D camera and 3-D lidar data to estimate the Time-to-Collision from the ego car to the preceding vehicle in a sequence of KITTI video data frames.

TTC Building Blocks

Course Structure

- **Lesson 3** : Keypoint detection and matching
- **Mid-Term Project** : Develop the matching framework and test several state-of-the-art algorithms.
- **Lesson 4** : Lidar point processing and deep learning for object detection.
- **Final Project** : Track 3D bounding boxes and compute refined TTC



This writeup walks through each rubric and includes a statement, supporting figures, and references to specific places in the code where each step is handled.

FP.1 MATCH BOUNDING BOXES

> *Implement the method "matchBoundingBoxes", which takes as input both the previous and 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.*

In this part of the project, we associated pairs of YOLO-identified bounding boxes across adjacent frames when they contained the highest number of matched keypoints relative to the other bounding box candidates on opposing frames.

Figure 1: YOLO Bounding Boxes on Frames 9 & 10

Figure 2: Keypoint Matches Between All Bounding Boxes on Frames 9 & 10 (Overlaps Filtered Out)

Figure 3: Keypoint Matches Between Bounding Boxes on the Preceding Vehicle

The implementation for matching and pairing bounding boxes across frames is shown below. Note that the matching keypoints between the images are stored in the data structure cv::DMatch such that queryIdx is the index of a keypoint in the previous frame, and trainIdx is the index of the corresponding keypoint match in the current frame.

Function Definition for Matching Bounding Boxes in camFusion_student.cpp:

```
void matchBoundingBoxes(std::vector<cv::DMatch> &matches, std::map<int, int> &bbBestMatches, DataFrame &prevFrame, DataFrame &currFrame)
{
    // count the number of keypoint matches that share the same bounding box between the current and previous frames
    cv::Mat count = cv::Mat::zeros(prevFrame.boundingBoxes.size(), currFrame.boundingBoxes.size(), CV_32S);
    for (auto matchpair : matches) // for each matched pair ...
    {
        // find the corresponding keypoint in the current and previous frame
        // note: kptMatches are ordered as follows: kptsPrev uses queryIdx and kptsCurr uses trainIdx
        cv::KeyPoint prevkp1 = prevFrame.keypoints.at(matchpair.queryIdx);
        cv::KeyPoint currkp1 = currFrame.keypoints.at(matchpair.trainIdx);

        for (int prevbb = 0; prevbb < prevFrame.boundingBoxes.size(); prevbb++)
        { // for each bounding box on the previous frame
            if (prevFrame.boundingBoxes[prevbb].roi.contains(prevkp1.pt))
            { // if the bounding box on the previous frame contains this keypoint
                for (int currbb = 0; currbb < currFrame.boundingBoxes.size(); currbb++)
                { // then loop through all bounding boxes on the current frame
                    if (currFrame.boundingBoxes[currbb].roi.contains(currkp1.pt))
                    { // if bounding box on the current frame also contains this keypoint
                        // then increment multimap counter for matches that share the same bounding box
                        count.at<int>(prevbb, currbb) = count.at<int>(prevbb, currbb) + 1;
                    }
                }
            }
        }
    } // end loop over all matched pairs between frames
```

```
    // associate bounding boxes that contain the highest number of matched pairs
    for (int prevBBIdx = 0; prevBBIdx < prevFrame.boundingBoxes.size(); prevBBIdx++)
    { // for each bounding box on the previous frame
        int boxID = -1, maxNumMatches = 0;
        for (int currBBIdx = 0; currBBIdx < currFrame.boundingBoxes.size(); currBBIdx++)
        { // for each bounding box on the current frame
            if (count.at<int>(prevBBIdx,currBBIdx) > maxNumMatches)
            { // find which pair of bounding boxes on prev and current frames have the maximum count
                boxID = currBBIdx;
                maxNumMatches = count.at<int>(prevBBIdx,currBBIdx);
            }
        }
        // save the box IDs of all matched pairs in bbBestMatches
        bbBestMatches[prevBBIdx] = boxID;
    }
}
```

Function Call in main():

```
/* +--------------------------------+ */
/* | TRACK 3D OBJECT BOUNDING BOXES | */
/* +--------------------------------+ */
// associate bounding boxes between current and previous frame using keypoint matches
map<int, int> bbBestMatches;
matchBoundingBoxes(matches, bbBestMatches, *(dataBuffer.end() - 2), *(dataBuffer.end() - 1));
// store best bounding box matches in current frame to the data buffer
(dataBuffer.end() - 1)->bbMatches = bbBestMatches;
```

FP.2 COMPUTE LIDAR-BASED TTC

> *Compute the time-to-collision for all matched 3D objects using only Lidar measurements from the matched bounding boxes between the current and previous frame. Filter out outlier Lidar points in a statistically robust way to avoid severe estimation errors.*

In this stage of the project, we worked to achieve accurate lidar-based measurements of the time-to-collision, starting with some fine-tuning. The following images show the lidar point cloud with the ego car at the origin. Measurements are available from a 360-degree, 3D scanning lidar sensor mounted on the roof of the ego car.

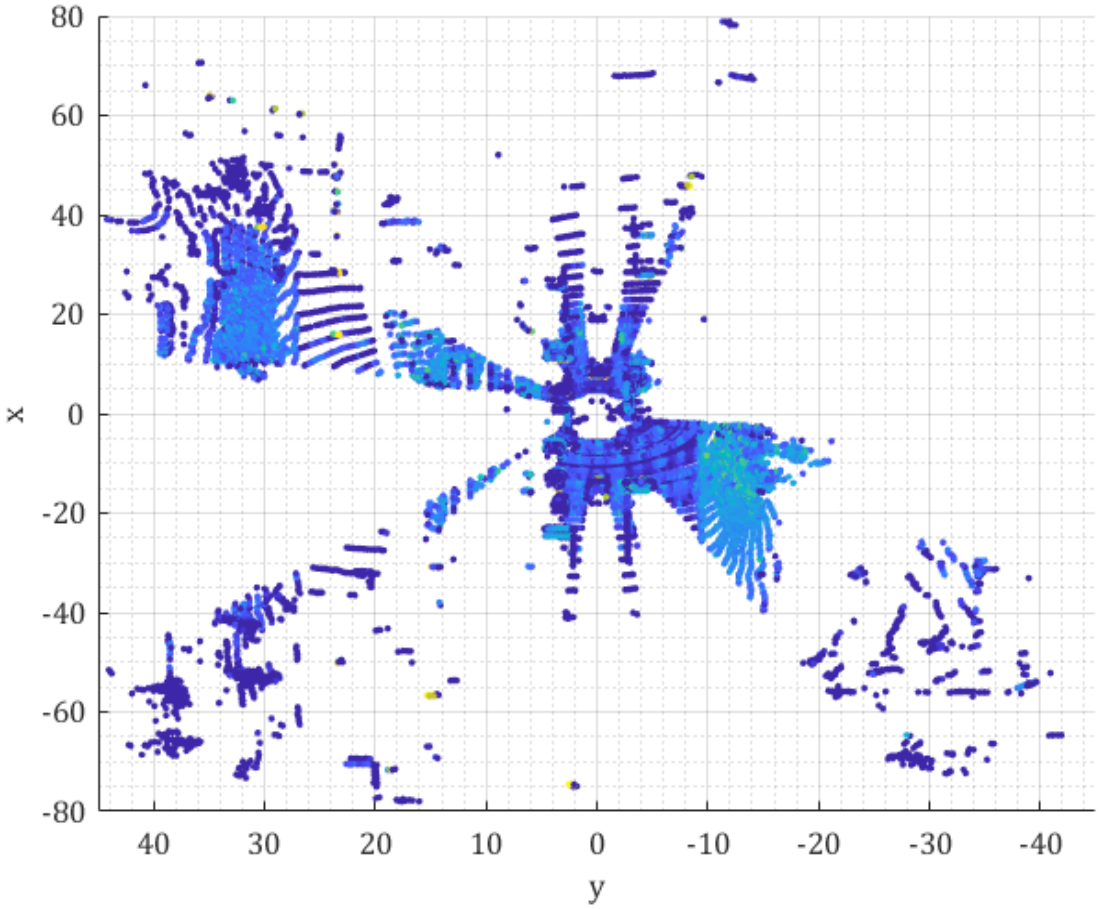


Figure 4: Top-Down View of Lidar Point Cloud

The region of interest across video frames was hand-tuned and cropped using a series of thresholds selected to exclude lidar points that were (1) outside of the ego lane, beyond 2 meters to the left or the right, (2) at or below ground level at -1.5 meters, (3) too far above the roof the car at -0.5 meters, (4) behind the ego car with negative x value, and (5) greater than 20 meters (too far away) in the direction of motion. Cropping the lidar points is crucial for this implementation. The following images show what happens if we do not perform this step:

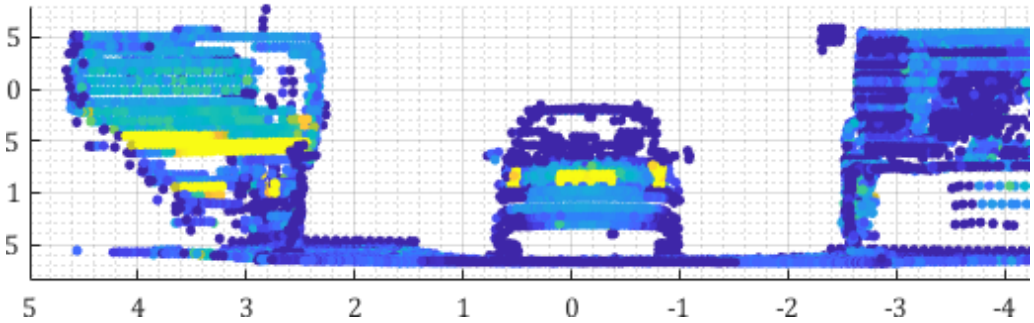


Figure 5: Forward-facing (x > 0) view of lidar point cloud in y-z world coordinates

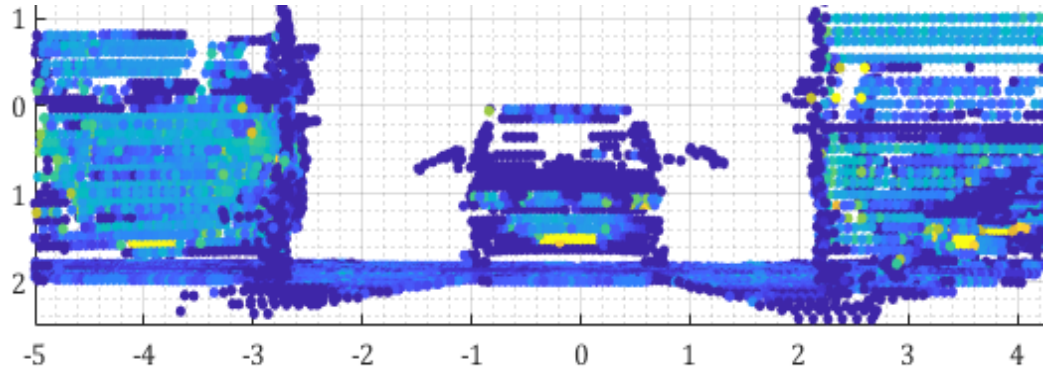


Figure 6: Backward-facing (x < 0) view of lidar point cloud in y-z world coordinates

The transformation from 3D world coordinates to 2D camera image results in the lidar points behind the vehicle forming an inverted image stacked above the scene in front of the vehicle:

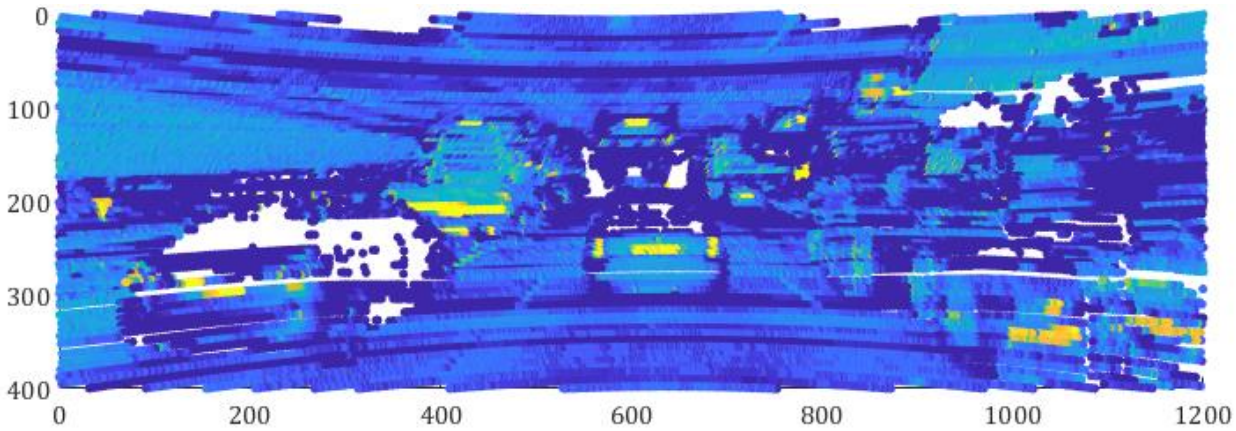


Figure 7: Full lidar point cloud mapped to 2D camera image plane, no cropping

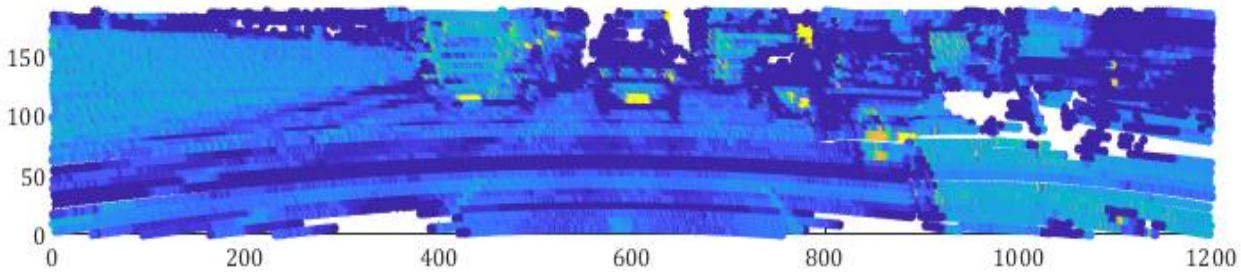


Figure 8: Inverted upper-half of uncropped 2D mapping shows the scene behind the ego car

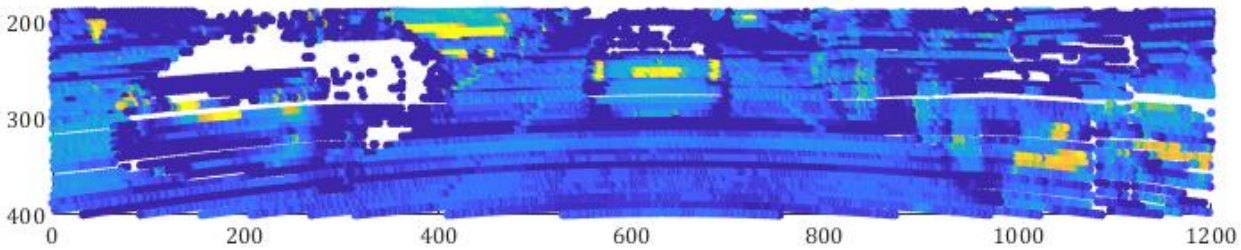


Figure 9: Bottom-half of uncropped 2D camera transformation contains the ego car's forward-facing view

Excluding negative x-values (and nothing else) produces the result shown below. The overlay of 2D-mapped lidar points stops approximately halfway up the image due to physical/hardware constraints in the lidar height and range.

Figure 10: Removed all points behind the ego car and mapped the result to 2D camera image frame

Next, we removed points from the road surface. A top-down view before and after performing this step:

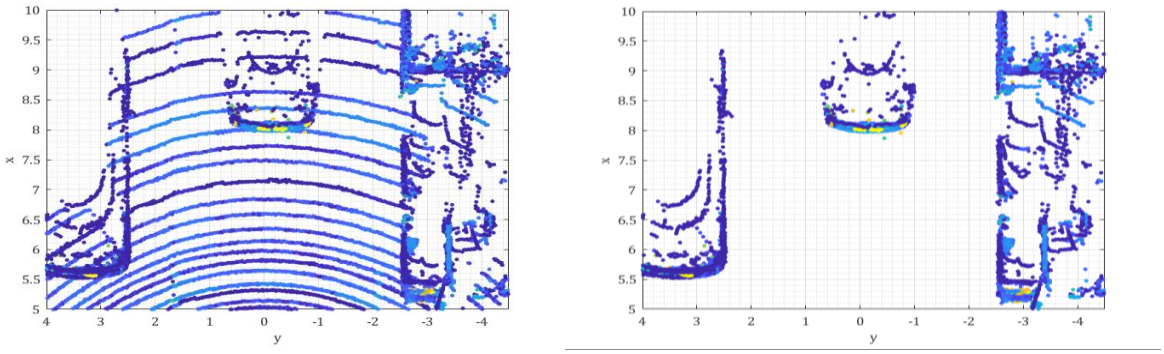


Figure 11: Removed measurements originating from the road surface

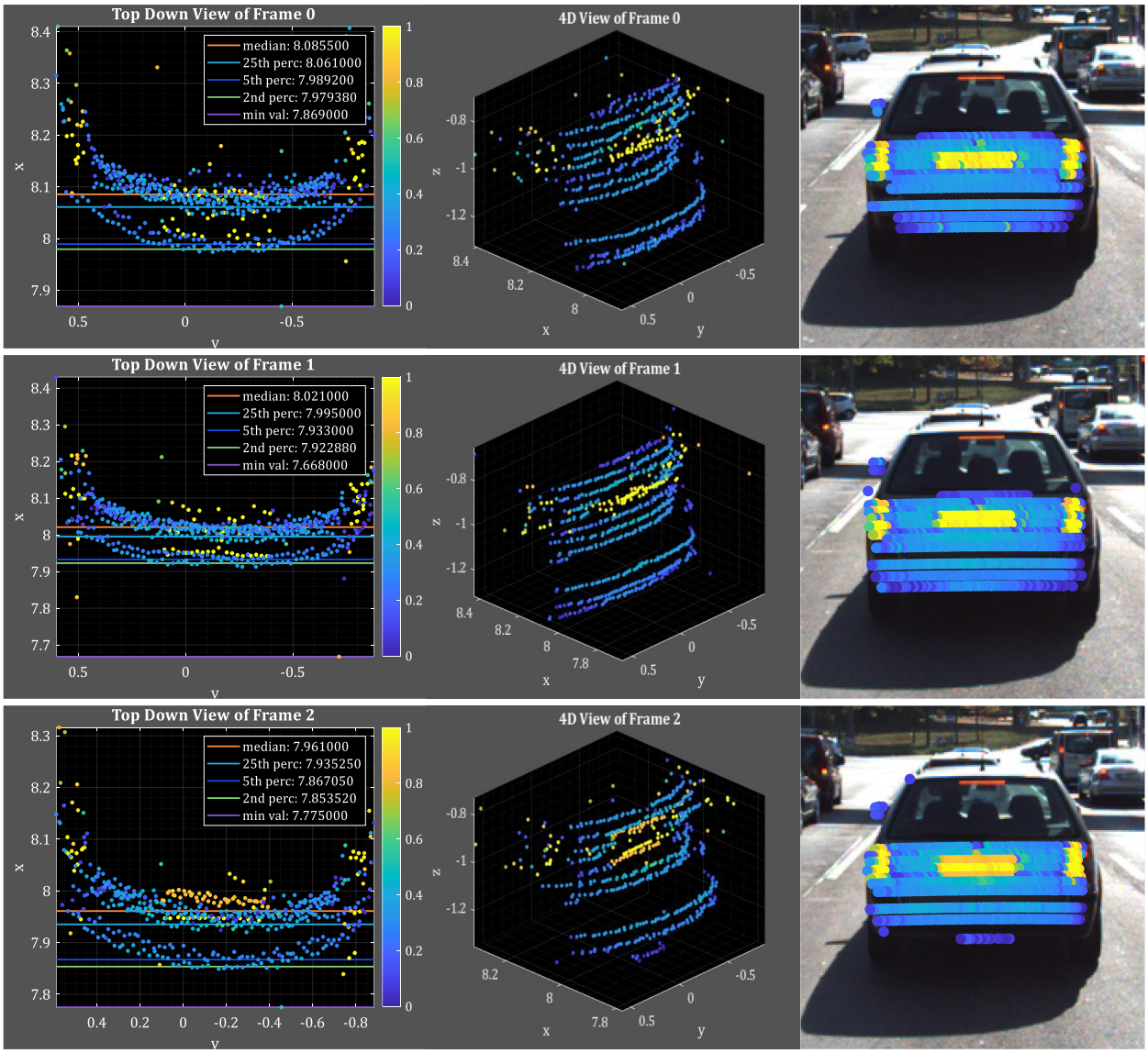
Finally, the image below shows the 2D overlay of lidar data after cropping out points according to the specifications described earlier: “not too high”, “not too low”, “not to far in front”, “not behind”, “not outside of the ego lane”.

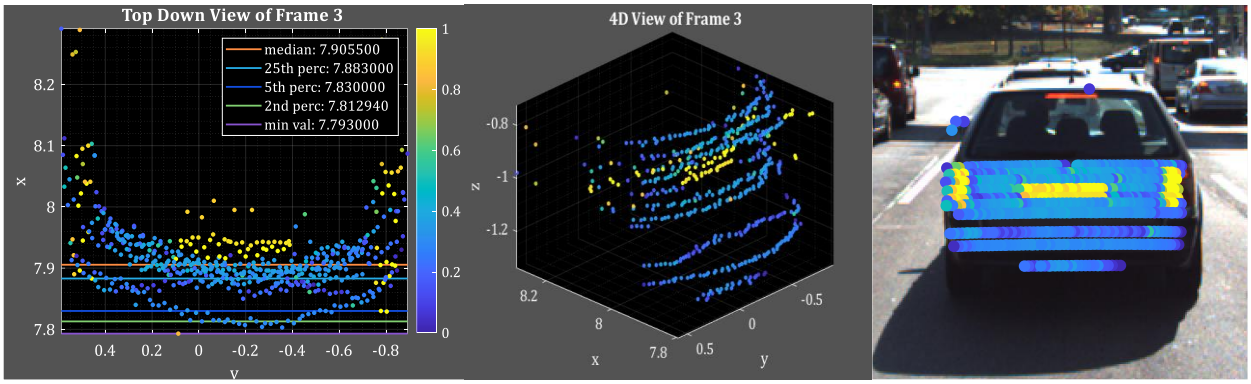
Figure 12: Cropped Lidar Point Cloud Mapped to 2D Camera Image Frame

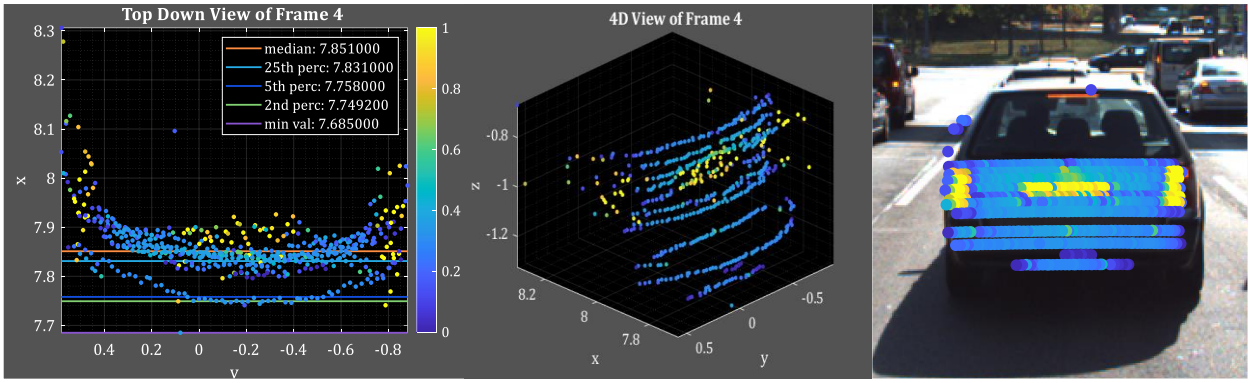
The TTC estimation space was further constrained by applying the (10% reduced) YOLO bounding box to the 2D mapped lidar points. This usually narrowed down the list to one bounding box:

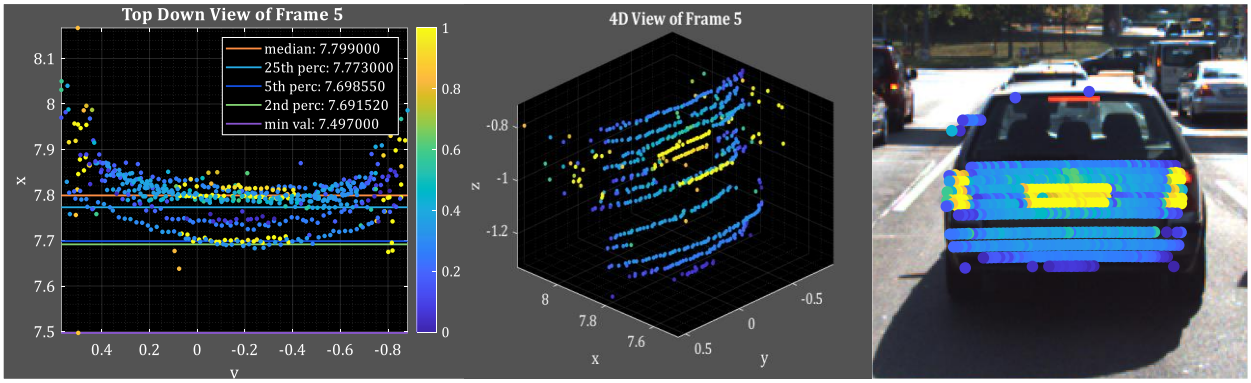
Figure 13: LIDAR Points Contained Within YOLO Bounding Box on Preceding Car (2-frame example)

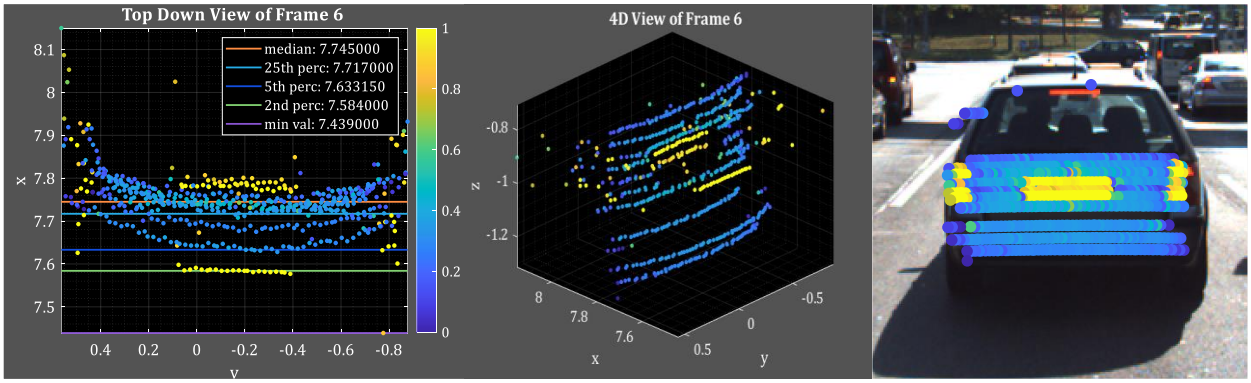
The distance to the preceding vehicle was estimated based on data within the region of interest. The following top-down views of the lidar measurements bouncing off the preceding vehicle show why the absolute minimum distances could not be used. Visual inspection and review of the percentiles across the distribution of x-distances were used to select “statistically robust” estimates in the presence of noise. In addition to x- and y-positions, the reflectivity of the data points was displayed according to the color scales shown next to each plot. The reflectivity information may show how variations in material properties (car paint, license plate, headlights, etc) and angle of incidence (rear-facing vs edges) contribute to noise in the position estimates.

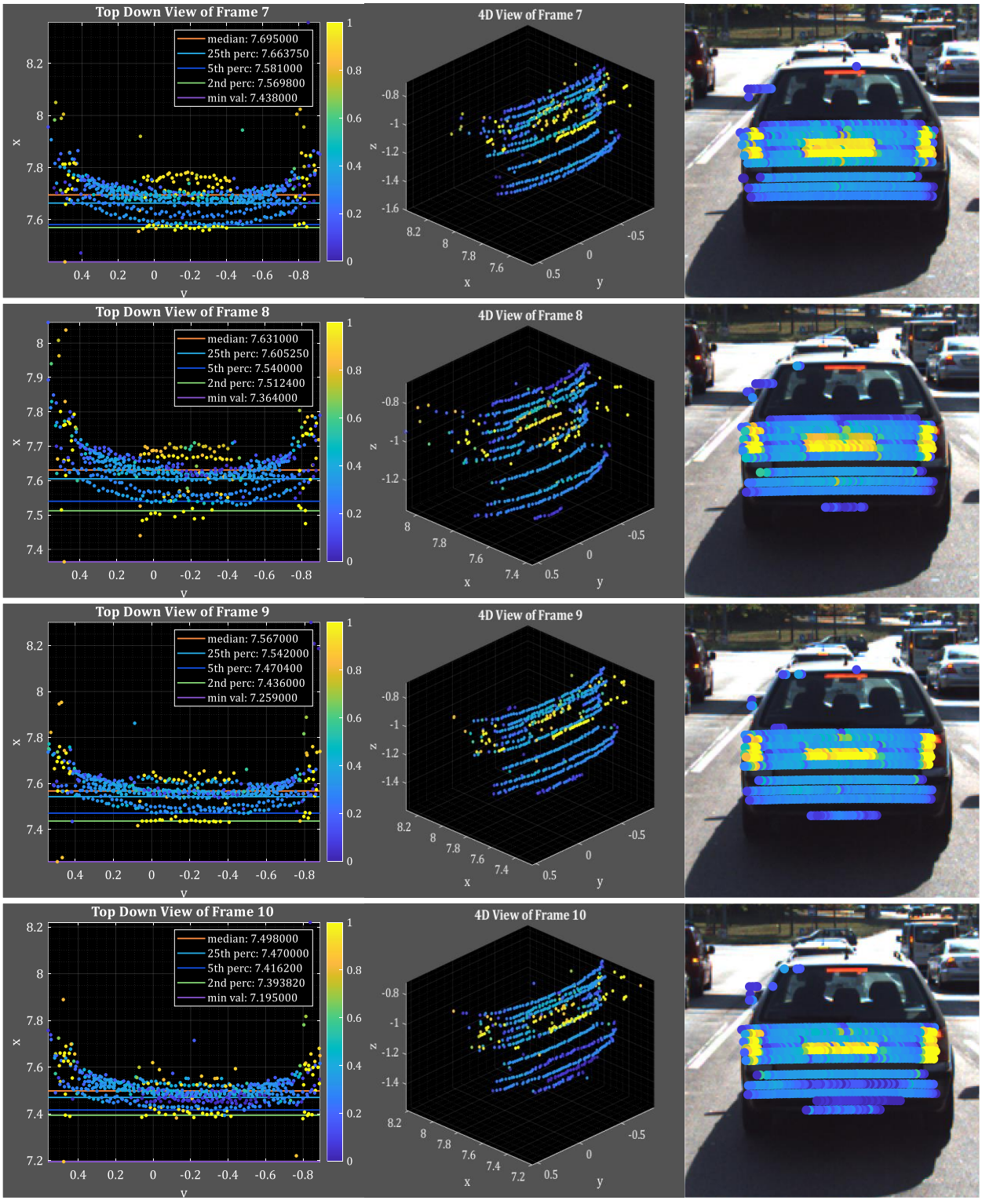


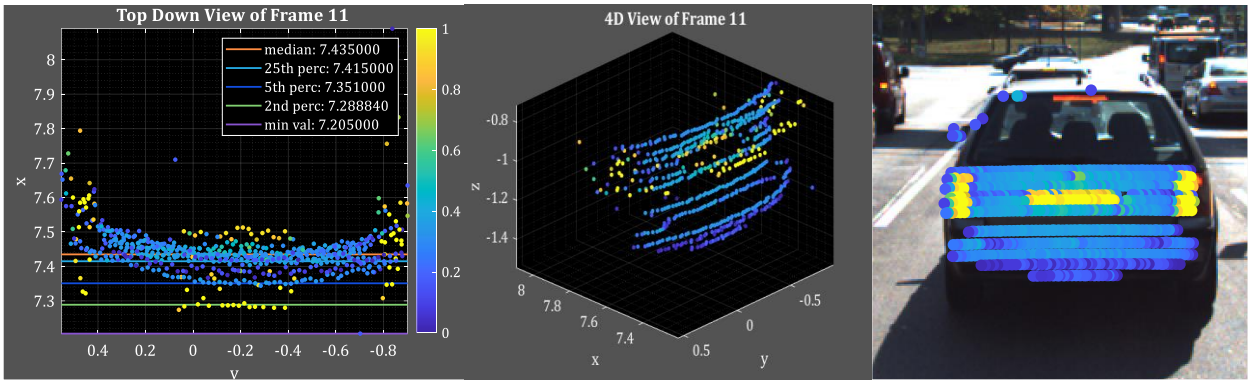

Top Down View of Frame 3
median: 7.905500
25th perc: 7.883000
5th perc: 7.830000
2nd perc: 7.812940
min val: 7.793000
x
y
4D View of Frame 3
x
y
z

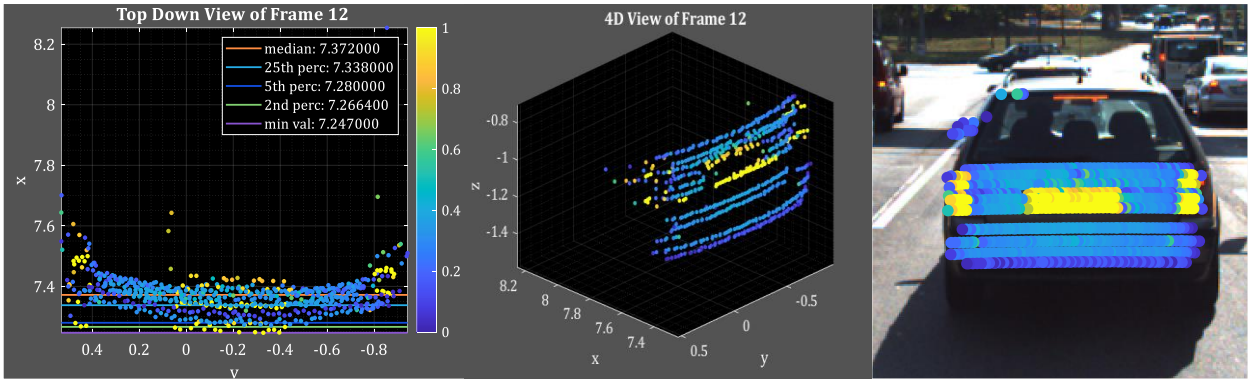

Top Down View of Frame 4
median: 7.851000
25th perc: 7.831000
5th perc: 7.758000
2nd perc: 7.749200
min val: 7.685000
x
y
4D View of Frame 4
x
y
z

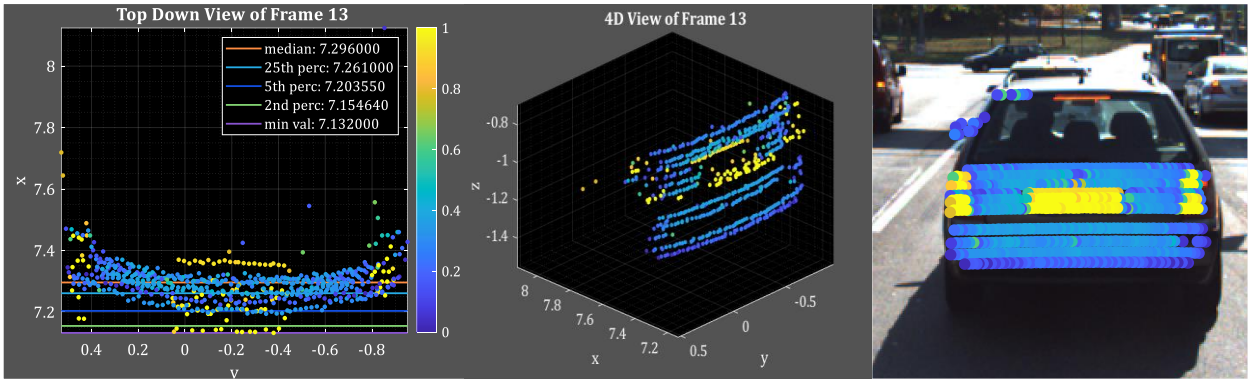

Top Down View of Frame 5
median: 7.799000
25th perc: 7.773000
5th perc: 7.698550
2nd perc: 7.691520
min val: 7.497000
x
y
4D View of Frame 5
x
y
z

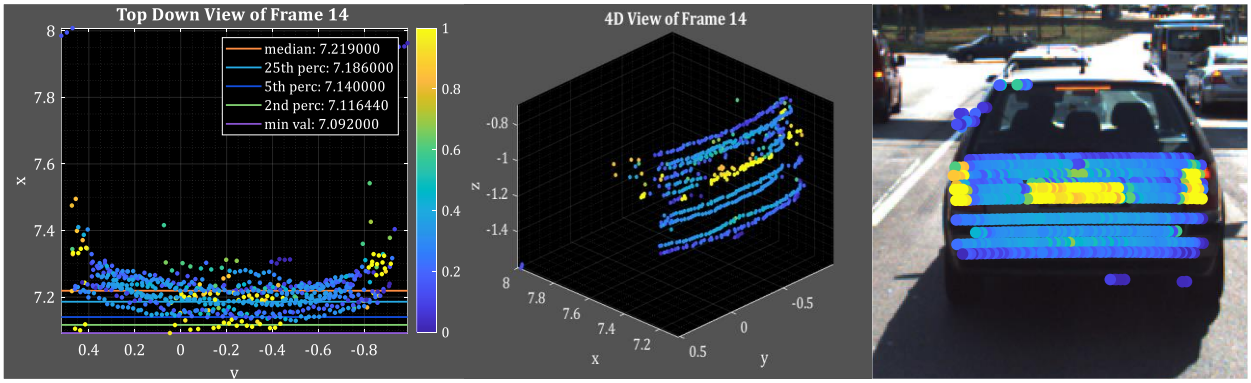

Top Down View of Frame 6
median: 7.745000
25th perc: 7.717000
5th perc: 7.633150
2nd perc: 7.584000
min val: 7.439000
x
y
4D View of Frame 6
x
y
z


Top Down View of Frame 7
median: 7.695000
25th perc: 7.663750
5th perc: 7.581000
2nd perc: 7.569800
min val: 7.438000
4D View of Frame 7
Top Down View of Frame 8
median: 7.631000
25th perc: 7.605250
5th perc: 7.540000
2nd perc: 7.512400
min val: 7.364000
4D View of Frame 8
Top Down View of Frame 9
median: 7.567000
25th perc: 7.542000
5th perc: 7.470400
2nd perc: 7.436000
min val: 7.259000
4D View of Frame 9
Top Down View of Frame 10
median: 7.498000
25th perc: 7.470000
5th perc: 7.416200
2nd perc: 7.393820
min val: 7.195000
4D View of Frame 10


Top Down View of Frame 11
median: 7.435000
25th perc: 7.415000
5th perc: 7.351000
2nd perc: 7.288840
min val: 7.205000
x
y
4D View of Frame 11
x
y
z


Top Down View of Frame 12
median: 7.372000
25th perc: 7.338000
5th perc: 7.280000
2nd perc: 7.266400
min val: 7.247000
x
y
4D View of Frame 12
x
y
z


Top Down View of Frame 13
median: 7.296000
25th perc: 7.261000
5th perc: 7.203550
2nd perc: 7.154640
min val: 7.132000
x
y
4D View of Frame 13
x
y
z


Top Down View of Frame 14
median: 7.219000
25th perc: 7.186000
5th perc: 7.140000
2nd perc: 7.116440
min val: 7.092000
x
y
4D View of Frame 14
x
y
z

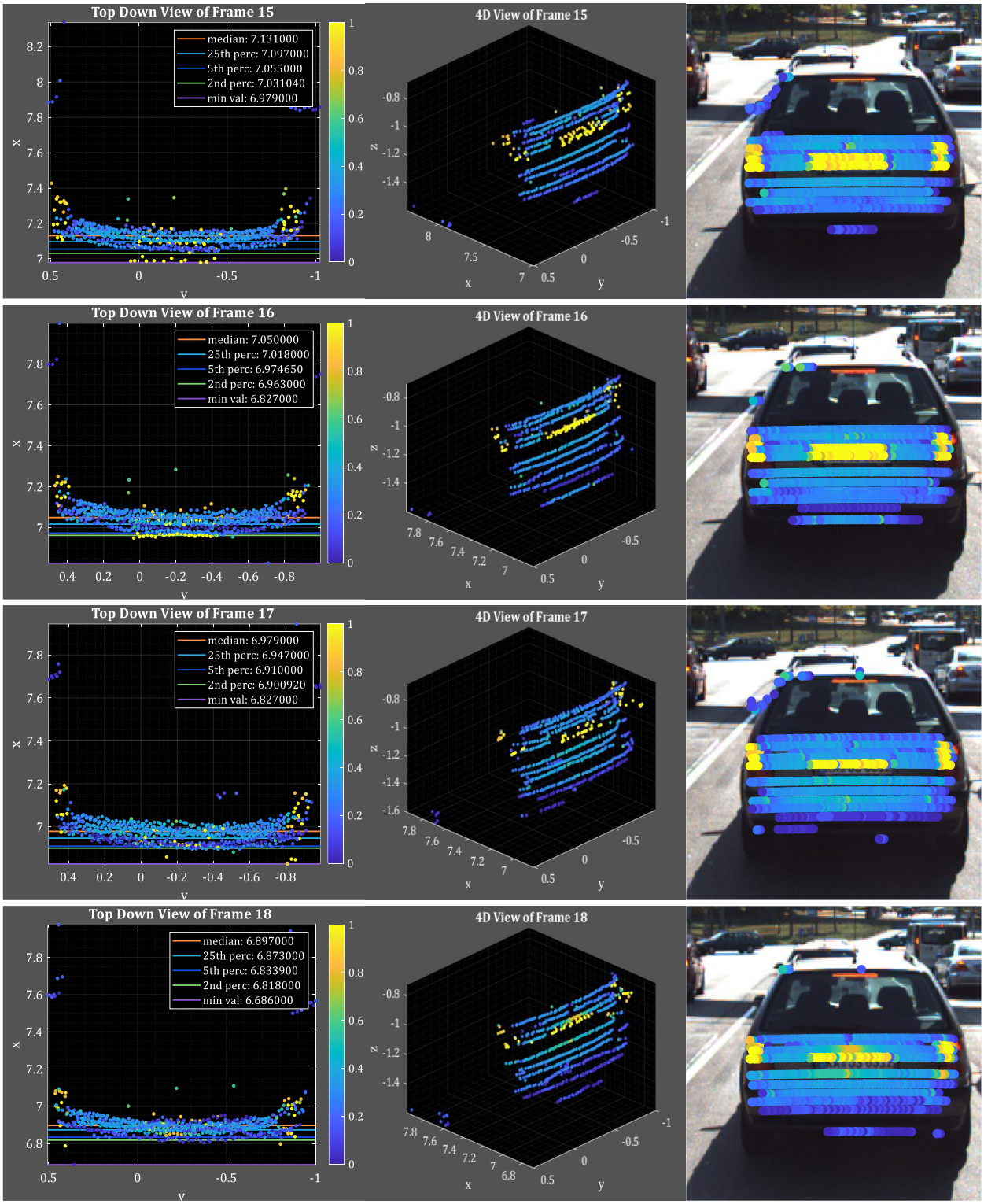


Figure 14: LIDAR point cloud views from different angles on each frame

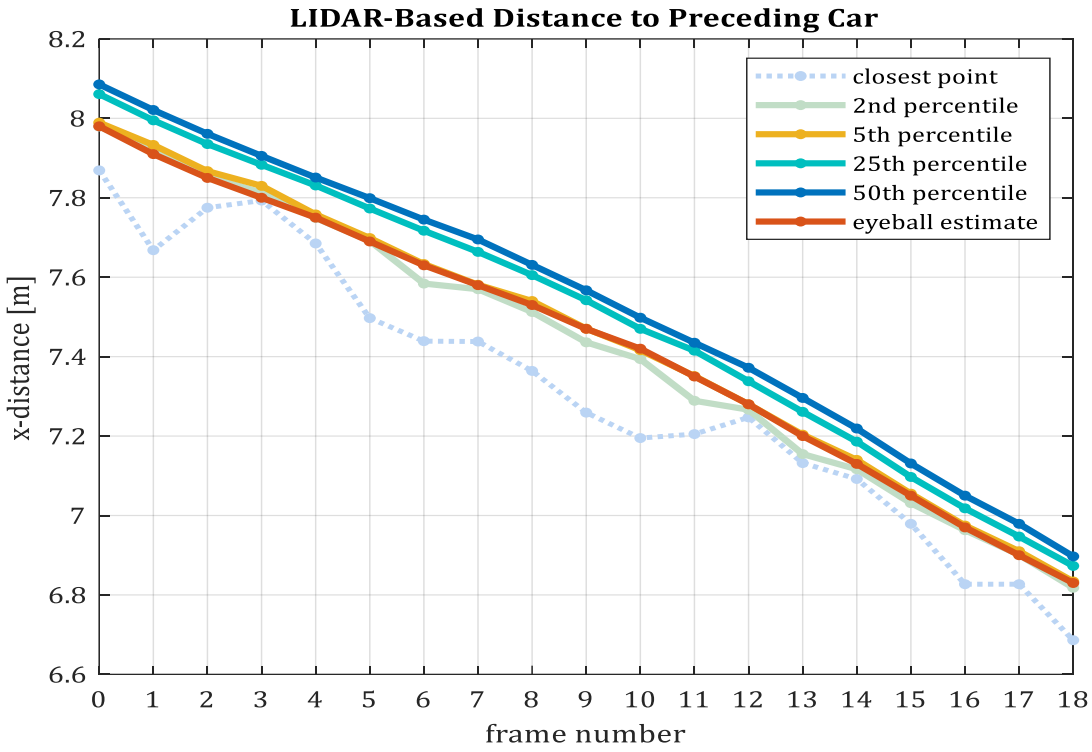


Figure 15: Distribution of lidar x-distances across frames

As discussed in the lecture notes, the ego car is equipped with a lidar sensor that can measure distances to obstacles at sequential time steps. For the purposes of estimating the time-to-collision, we are interested in the distances to the closest 3D point in the path of driving, illustrated here as distance d_0 at time 1, and distance d_1 at time 2:

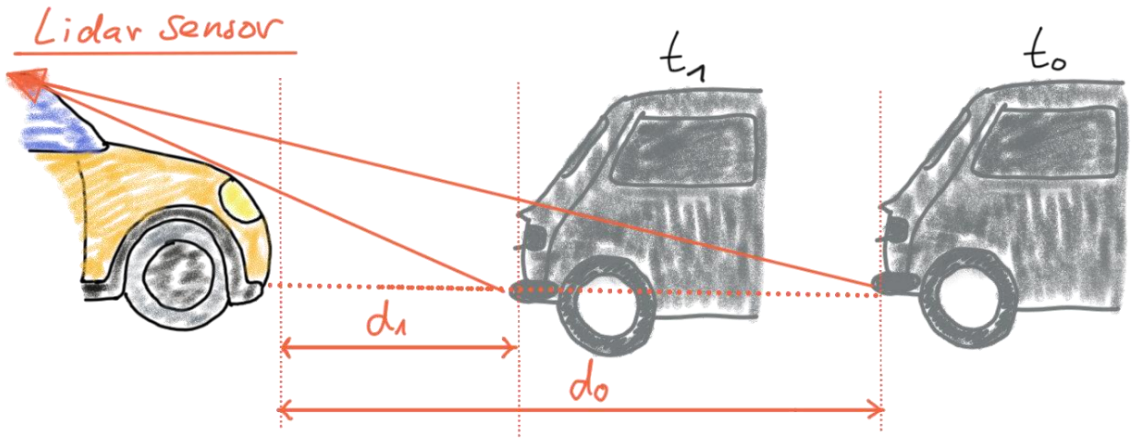


Figure 16: Illustration of the variables needed to compute the lidar-based time-to-collision

Based on the model of constant velocity, we can compute the time-to-collision between two successive lidar measurements as follows:

$$TTC = \frac{d_1 * \Delta t}{d_0 - d_1}$$

Note that the distances d_0 and d_1, are measured from the front end of the ego car to the back end of the preceding car, not from the roof-mounted lidar sensor to the preceding car. There is an extrinsic calibration from the sensor position to the front-end position of the ego car that needs to be computed. Since we do not have the coefficients for this transformation, I used an "eyeball" estimate of 4.6m for this offset based on a top-down view of when we start to see the road below the front of the vehicle:

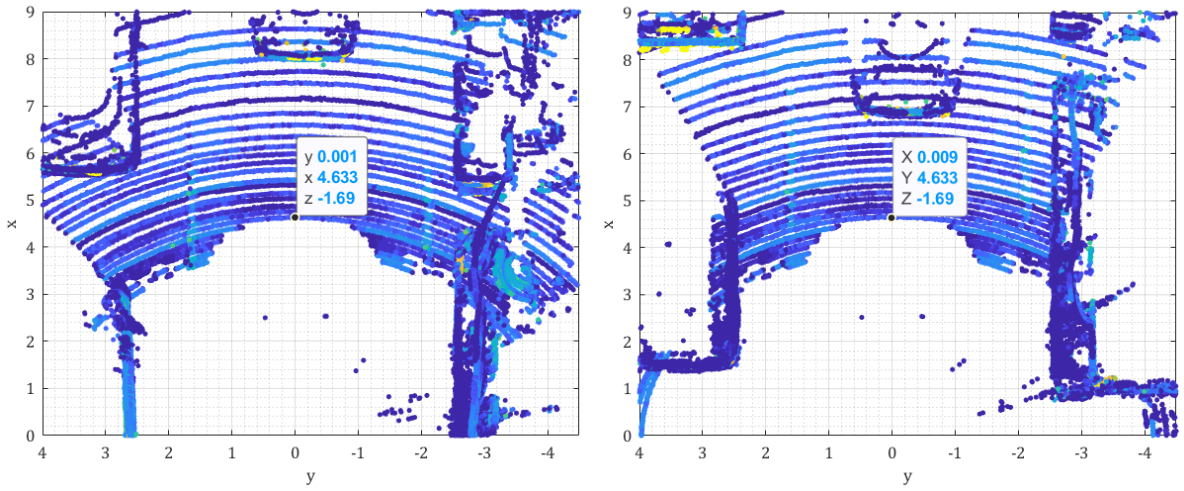


Figure 17: Estimated distance from the roof-mounted lidar sensor to the front-edge of ego car: Frames 0 and 18

Puzzlingly, subtracting an offset of 4.6m from all measured x-distances produced lidar-based TTC estimates that were about 4.6m too small relative to the camera-based TTC estimates (described later in this report). In the absence of truth, I relied on such similarities to cross-validate the implementations. For the purposes of this project, I decided not to subtract the offset from the x-distance measurements. The following graph below shows the TTC estimates *without* the lidar-to-edge distance compensation.

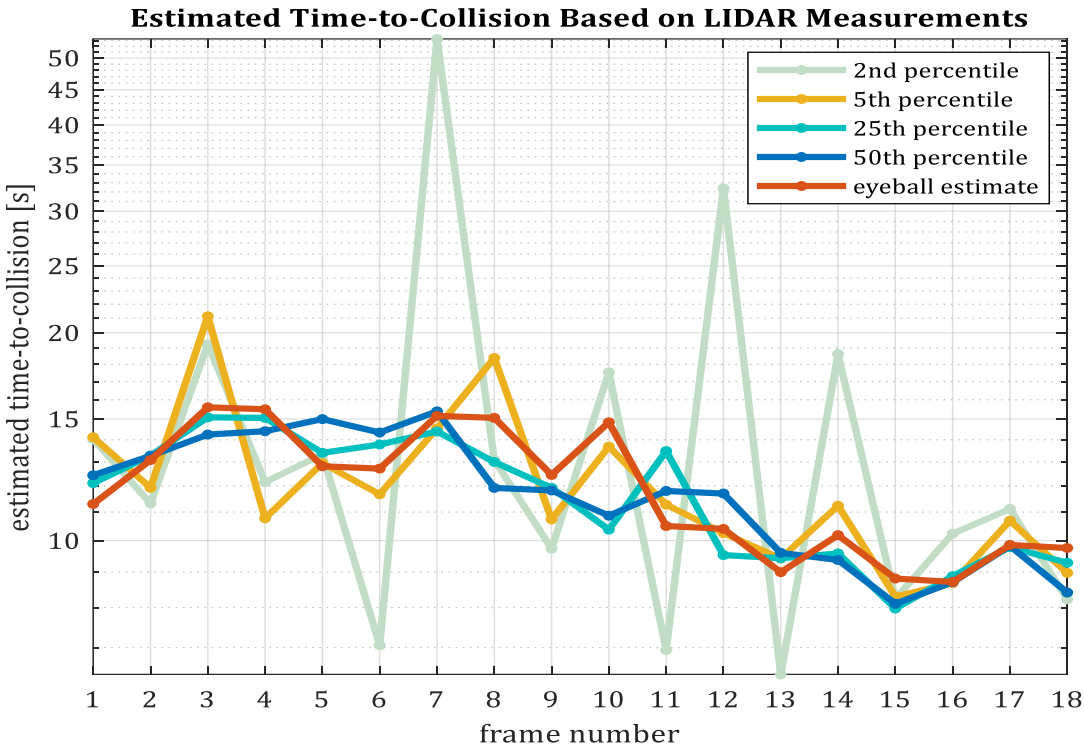


Figure 18: Lidar-based TTC estimates computed at percentiles of the x-distance distribution in the region of interest

Fluctuations in the lidar-based TTC estimates over time demonstrated a high-sensitivity to noise, as shown in the figures above. Based on this graph, it seems that the 25th and 50th raw percentiles provide the most stable frame-to-frame estimates, at the expense of slightly over-estimating the time-to-collision. A temporal smoothing approach could potentially enable us to choose a percentile that runs closer to the back end of the vehicle. For the purposes of this quick-study, I decided to forego implementation of a temporal smoothing filter and selected the x-position corresponding to the 25th percentile of the data set.

Function Definition of LIDAR-based Time-to-Collision in camFusion_student.cpp:

```cpp
// Compute the time-to-collision for all matched 3D objects based on lidar measurements alone
void computeTTCLidar(std::vector<LidarPoint> &lidarPointsPrev,
                     std::vector<LidarPoint> &lidarPointsCurr, double frameRate, double &TTC)
{
    // reformat x (distance) values as a vector of doubles
    std::vector<double> distPrev;
    std::vector<double> distCurr;

    for (auto it = lidarPointsPrev.begin(); it != lidarPointsPrev.end(); ++it)
        distPrev.push_back(it->x);

    for (auto it = lidarPointsCurr.begin(); it != lidarPointsCurr.end(); ++it)
        distCurr.push_back(it->x);

    // take nth smallest point as the distance to use (anything smaller is discarded as noise)
    int nthP, nthC;
    float prctile = 0.25; //0.5; // nth point is based on a fraction of the total number of points
    nthP = floor(prctile * distPrev.size());
    std::nth_element(distPrev.begin(), distPrev.begin() + (nthP - 1), distPrev.end());
    nthC = floor(prctile * distCurr.size());
    std::nth_element(distCurr.begin(), distCurr.begin() + (nthC - 1), distCurr.end());

    // based on the model of constant velocity, the TTC can be computed from two successive lidar measurements as follows:
    TTC = (distCurr[nthC - 1] * (1 / frameRate)) / (distPrev[nthP - 1] - distCurr[nthC - 1]);
}
```

Function Call in main():

```cpp
    // if the bounding boxes on the curr and prev frames have lidar points associated with them
    // (note: lidar points were previously cropped to capture obstacles only in the ego lane)
    if (currBB->lidarPoints.size() > 0 && prevBB->lidarPoints.size() > 0)
    {
        double ttcLidar = -1;
        // then compute time-to-collision based on LIDAR data for the current match
        computeTTCLidar(prevBB->lidarPoints, currBB->lidarPoints, sensorFrameRate, ttcLidar);
```

FP.3 ASSOCIATE KEYPOINT MATCHES WITH BOUNDING BOXES

> *Prepare the TTC computation based on camera measurements by associating keypoint correspondences to the bounding boxes that enclose them. All matches that satisfy this condition must be added to a vector representing the respective bounding box. Remove outlier matches based on the Euclidean distance between them in relation to all matches in the bounding box.*

In this section, we computed the Euclidean distance between all keypoints within the bounding box (usually one, but sometimes two) on the preceding vehicle, which were identified with the help of the highly-constrained lidar space defined in the previous section. “Outlier matches” were identified and removed, and the selected matched keypoints were stored in a dynamic list associated with the respective bounding box structures.

I experimented with 2 different methods for removing outlier matches: (1) RANSAC and (2) an empirically-determined threshold multiplier applied to the mean distance between keypoint matches. Statistically, the approaches produced comparable TTC estimates with reasonable repeatability across frames. However, a cursory visual inspection of both approaches showed that RANSAC did a better job of identifying out “bad” keypoint matches on the road, surrounding objects or shadows.

Earlier implementations of this approach showed significant errors in the estimated TTC due to keypoints on adjacent cars, the road, shadows, or other elements that fell within the bounding box but did not physically belong to the preceding car. To mitigate these effects, I added the following pre-filtering steps:

1) Excluding keypoints that did not belong to any bounding box
2) Removing keypoints that fell within more than one YOLO bounding box
3) Shrinking the bounding box slightly to avoid having too many outlier points around the edges of the vehicle

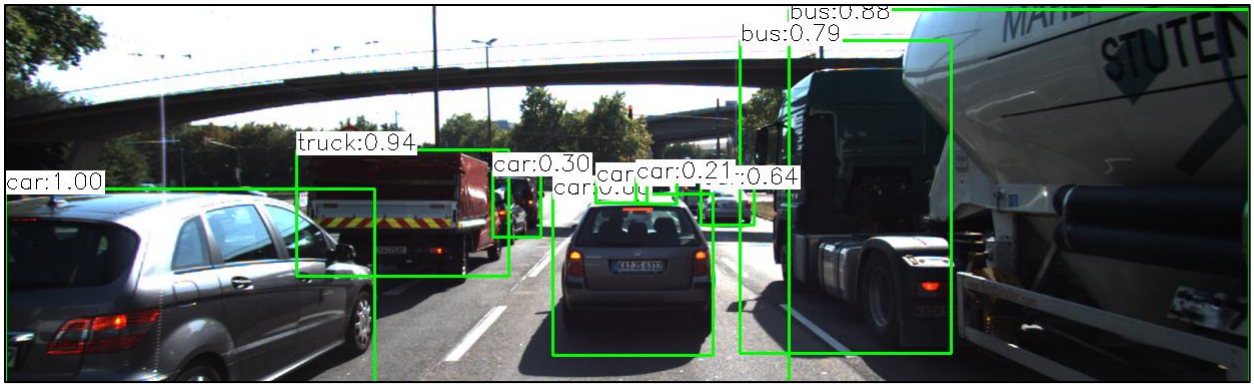


Figure 19: YOLO Bounding Boxes on Frame 7

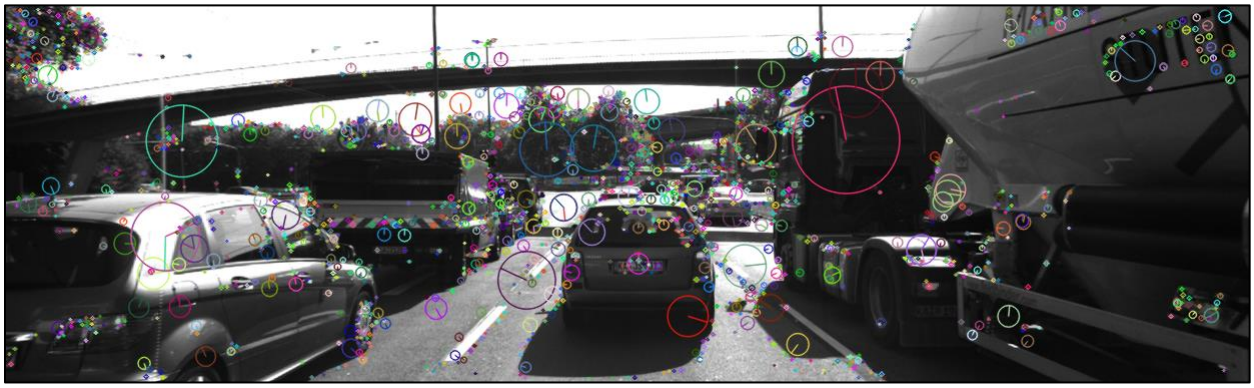


Figure 20: Original SIFT Keypoints on Frame 7

Figure 21: SIFT Keypoints Filtered to Non-Overlapping Bounding Boxes

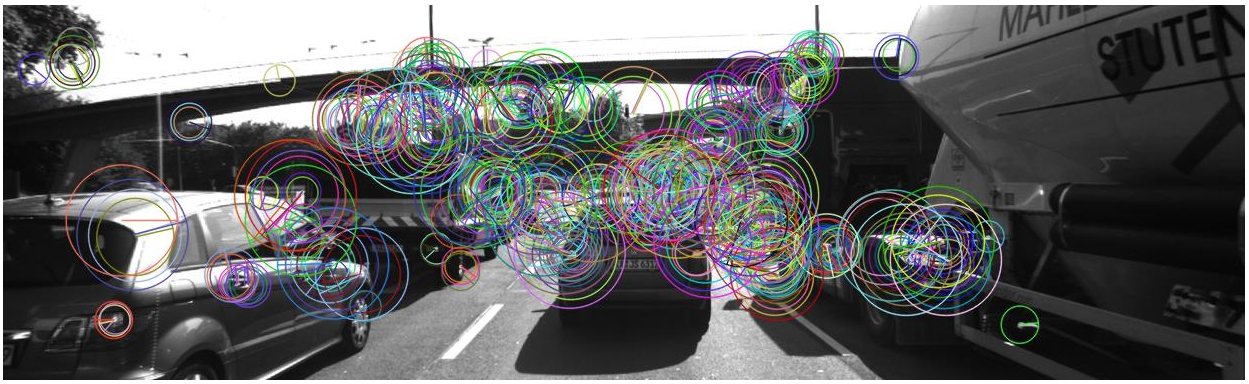


Figure 22: Original ORB Keypoints on Frame 7

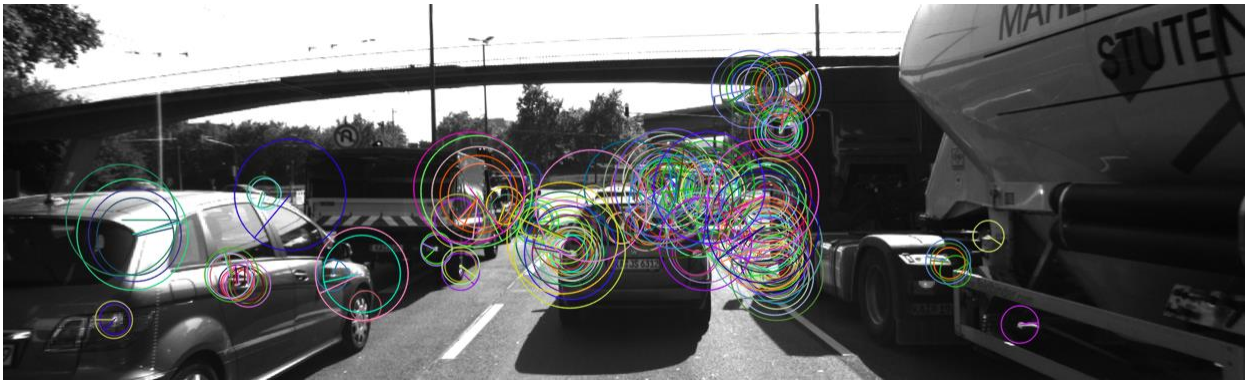


Figure 23: ORB Keypoints Filtered to Non-Overlapping Bounding Boxes

Figure 24: Original Shi-Tomasi Keypoints on Frame 7

Figure 25: Shi-Tomasi Keypoints Filtered to Non-Overlapping Bounding Boxes

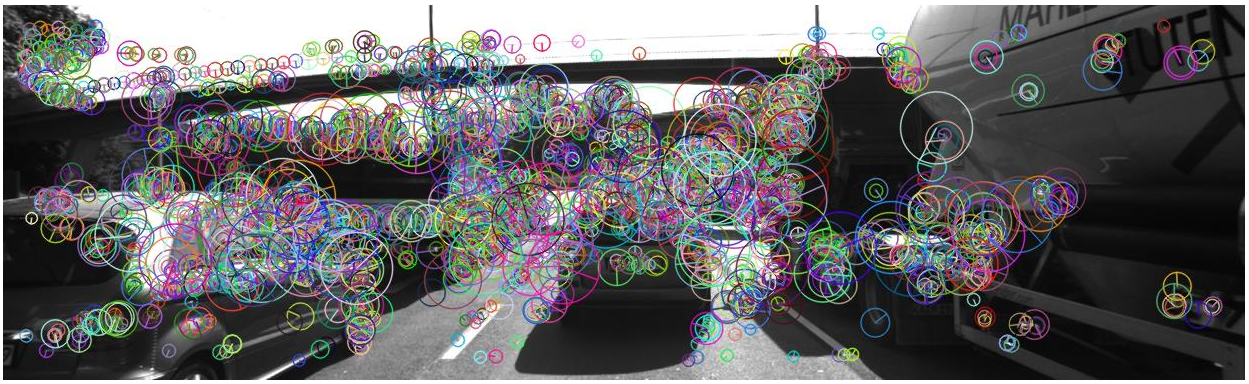


Figure 26: Original BRISK Keypoints on Frame 7

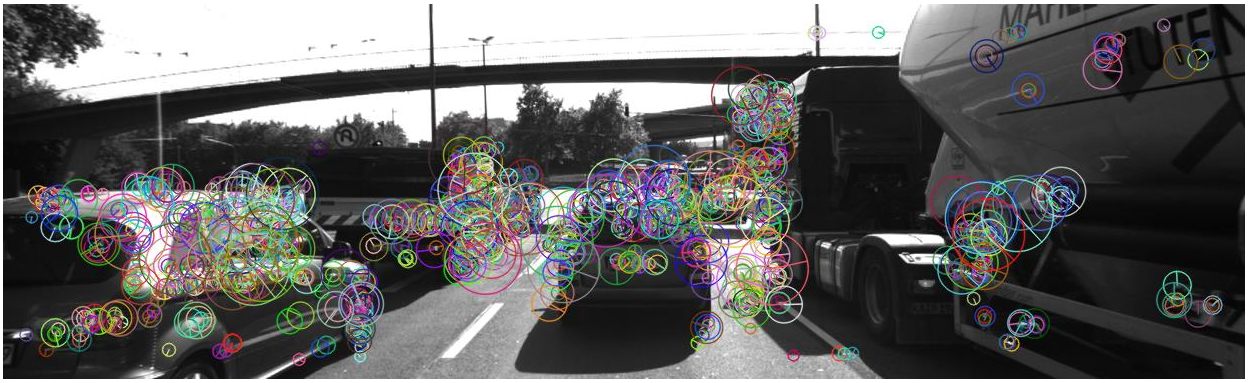


Figure 27: BRISK Keypoints Filtered to Non-Overlapping Bounding Boxes

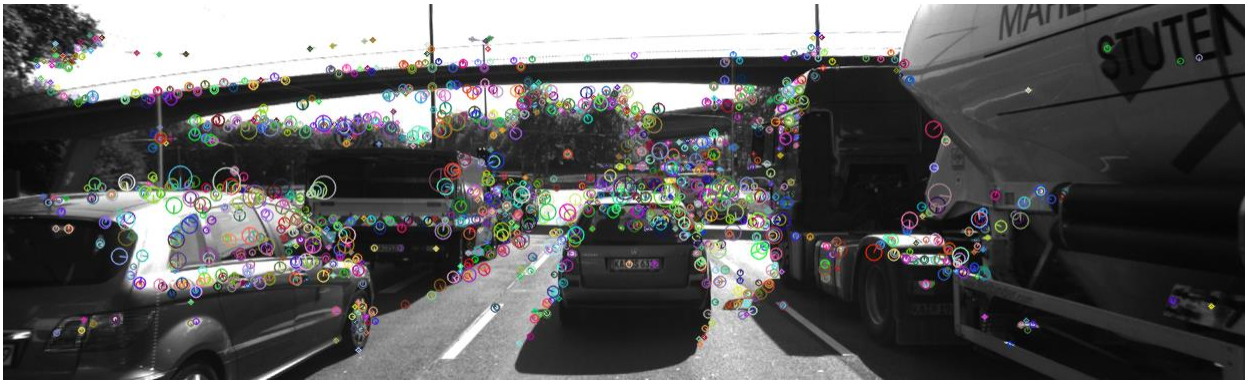


Figure 28: Original AKAZE Keypoints on Frame 7

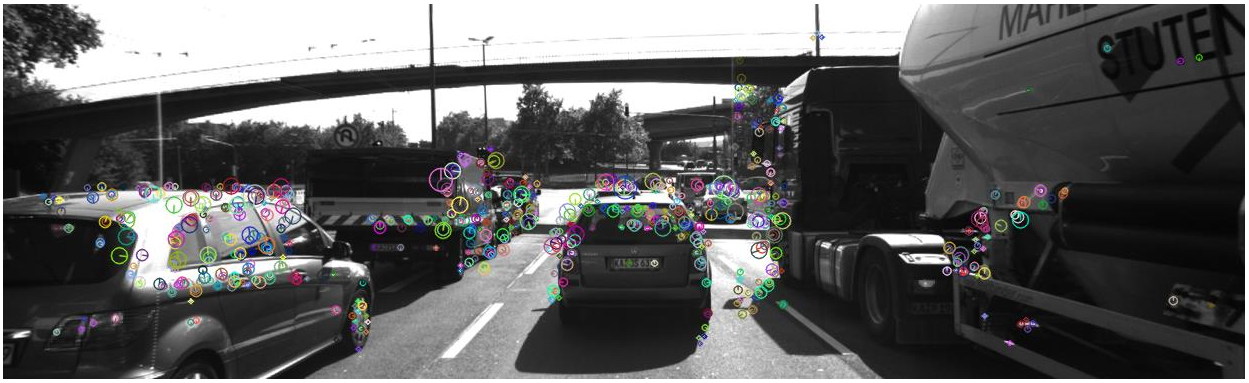


Figure 29: AKAZE Keypoints Filtered to Non-Overlapping Bounding Boxes

Function Definition to Reduce Keypoints to Non-Overlapping Bounding Boxes:

```
std::vector<cv::KeyPoint> eraseKptsInOverlappingBBs(std::vector<BoundingBox> &boundingBoxes,
                                                    std::vector<cv::KeyPoint> &keypoints)
{
    std::vector<cv::KeyPoint> filteredKeypoints;
    // loop over all keypoints and associate them to a 2D bounding box
    for (auto currKpt : keypoints)
    {
        int nBBsContainingKpt = 0;
        for (auto it2 = boundingBoxes.begin(); it2 != boundingBoxes.end(); ++it2)
        {
            // shrink current bounding box slightly to avoid having too many outlier points around the edges
            cv::Rect smallerBox;
            smallerBox.x = it2->roi.x;
            smallerBox.y = it2->roi.y;
            smallerBox.width = it2->roi.width;
            smallerBox.height = it2->roi.height;

            // check whether point is within current bounding box
            if (smallerBox.contains(currKpt.pt))
                nBBsContainingKpt++;

        } // end loop over all bounding boxes

        // in order to avoid inadvertent association of a keypoint on one vehicle with another vehicle,
        // exclude keypoints that are enclosed within multiple bounding boxes from further processing
        if (nBBsContainingKpt == 1) // if keypoint is enclosed by exactly one bounding box
            filteredKeypoints.push_back(currKpt);

    } // end loop over all keypoints
    return filteredKeypoints;
}
```

The following images are examples of keypoint matches associated to the nearest ego lane bounding box between frames, where red markers represent keypoints on the previous frame, and green markers represent the corresponding keypoint matches on the current frame:

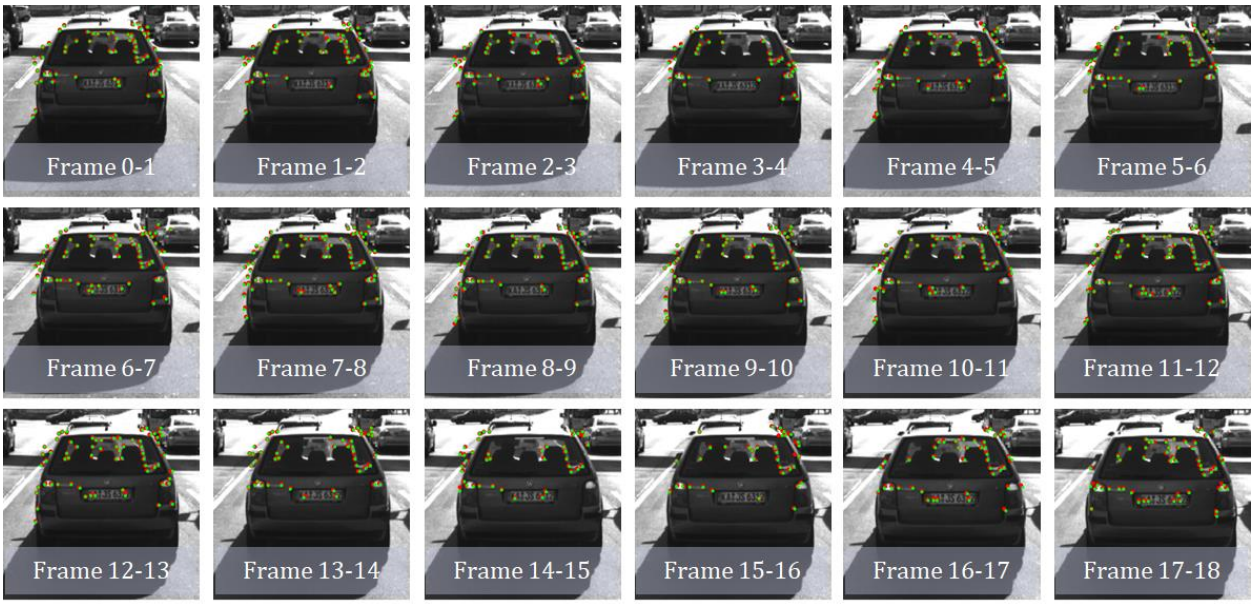


Figure 30: Frame-to-Frame Bounding Box Matches Using Shi-Tomasi Detector - SIFT Descriptor

Function Definition to Associate Keypoint Matches to Bounding Boxes:

```cpp
// associate a given bounding box with the keypoints it contains
void clusterKptMatchesWithROI(BoundingBox &boundingBoxPrev, BoundingBox &boundingBox,
                              std::vector<cv::KeyPoint> &kptsPrev, std::vector<cv::KeyPoint> &kptsCurr,
                              std::vector<cv::DMatch> &kptMatches)
{
    // init variables for mean-based match selection
    double meanDist = 0.0, sumDist = 0.0;
    int numGoodMatches = 0;
    std::map<cv::DMatch, double> bbMatches;
    std::vector<cv::KeyPoint> tmpKpts, tmpKptsPrev;

    // init variables to find good matches using RANSAC
    std::vector<cv::Point2f> pointsPrev, pointsCurr;
    std::vector<cv::DMatch> bbMatches2;

    for (auto match : kptMatches)
    {
        // shrink current bounding box slightly to avoid having too many outlier points around the edges
        cv::Rect smallerBox;
        smallerBox.x = boundingBox.roi.x + 0.05 * boundingBox.roi.width / 2.0;
        smallerBox.y = boundingBox.roi.y + 0.05 * boundingBox.roi.height / 2.0;
        smallerBox.width = boundingBox.roi.width * (1 - 0.01);
        smallerBox.height = boundingBox.roi.height * (1 - 0.2);

        // if the smaller bounding box contains the current keypoint in the current match
        // note: kptMatches are ordered as follows: kptsPrev uses queryIdx and kptsCurr uses trainIdx
        if (smallerBox.contains(kptsCurr[match.trainIdx].pt))
        {
            // convert keypoints to Point2f (for RANSAC)
            pointsPrev.push_back(cv::Point2f(kptsPrev[match.queryIdx].pt.x, kptsPrev[match.queryIdx].pt.y));
            pointsCurr.push_back(cv::Point2f(kptsCurr[match.trainIdx].pt.x, kptsCurr[match.trainIdx].pt.y));

            // save keypoints within the bounding box
            tmpKpts.push_back(kptsCurr[match.trainIdx]);
            tmpKptsPrev.push_back(kptsPrev[match.queryIdx]);

            // compute the distance between the previous and current positions of the keypoint
            double distance = cv::norm(kptsCurr[match.trainIdx].pt - kptsPrev[match.queryIdx].pt);
            bbMatches[match] = distance; // save distance to match structure
            bbMatches2.push_back(match);
            sumDist += distance; // keep a running sum of distances between mapped keypoint positions
            numGoodMatches++;    // keep a counter of number of good keypoint matches
        }
    }
```

Method 1 uses RANSAC to identify & select “good” keypoint matches:

```cpp
    bool useRansac = true;
    if (useRansac)
    {
        // Compute the homography matrix and inlier (good) matches using RANSAC
        std::vector<uchar> inliers; //cv::Mat inliers;
        cv::Mat H = cv::findHomography(
            cv::Mat(pointsPrev), // matching points in previous frame
            cv::Mat(pointsCurr), // matching points in current frame
            cv::RANSAC,          // RANSAC method
            3,                   // RANSAC reprojection threshold
            inliers);            // output mask where 0s indicate outliers

        // select only the inliers
        for (int i = 0; i < inliers.size(); i++)
        {
            if ((int)inliers.at(i))
            {
                boundingBox.kptMatches.push_back(bbMatches2[i]);
                boundingBoxPrev.kptMatches.push_back(bbMatches2[i]);

                boundingBox.keypoints.push_back(tmpKpts[i]);
                boundingBoxPrev.keypoints.push_back(tmpKptsPrev[i]);
            }
        }
    }
```

Method 2 applies a threshold multiplier to the mean distance:

```
    else
    {
        // filter out matches that are not within a threshold of the mean match distance
        int kptNum = 0;
        float outlierThresh = 2.5; // empirically selected threshold
        meanDist = sumDist / numGoodMatches; // compute the average distance
        for (auto matchDist : bbMatches)     // loop over all bounding box matches
        {
            if (matchDist.second < outlierThresh * meanDist) // filter out outliers
            {
                boundingBox.kptMatches.push_back(matchDist.first);
                boundingBoxPrev.kptMatches.push_back(matchDist.first);

                boundingBox.keypoints.push_back(tmpKpts[kptNum]);
                boundingBoxPrev.keypoints.push_back(tmpKptsPrev[kptNum]);
            }
            kptNum++;
        }
    }
}
```

Function Call in main():

```
    // assign enclosed keypoint matches to the bounding box
    clusterKptMatchesWithROI(*prevBB, *currBB,
                             (dataBuffer.end() - 2)->keypoints,
                             (dataBuffer.end() - 1)->keypoints,
                             (dataBuffer.end() - 1)->kptMatches);
```

FP.4 COMPUTE CAMERA-BASED TTC

> *Compute the Time-to-Collision for all matched 3D objects using only keypoint correspondences from the matched bounding boxes between the current and previous frames.*

A mono camera can estimate the time-to-collision via perspective projection by observing relative changes in relative height (a.k.a. scale change) directly in the image.

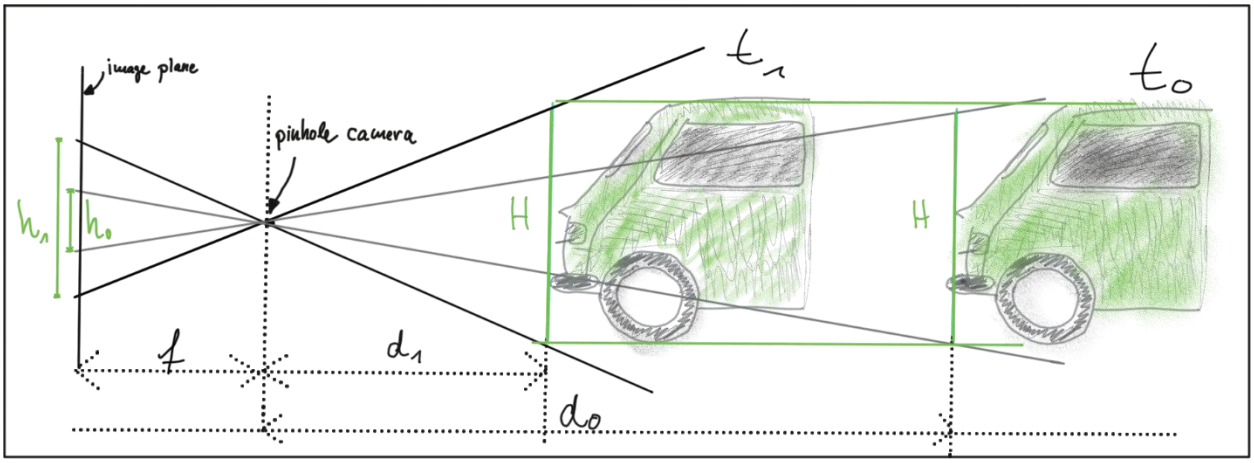


Figure 31: Estimating the Time-to-Collision Using a Single Camera

As was described in the lecture notes, we can substitute the relative distances of the keypoints captured in two temporally separated snapshots of an object moving forward or backward in front of and relative to our ego car to estimate the TTC from a single monocular camera:

$$TTC = \frac{-\Delta t}{\left(1 - \frac{h_1}{h_0}\right)}$$

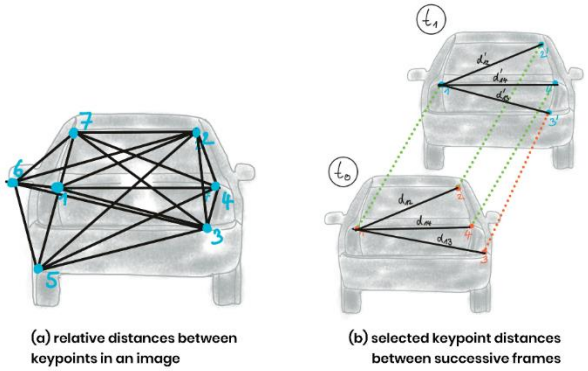


Figure 32: Using Relative Distances Between Texture-Based Keypoints to Estimate the TTC

We bound the problem to a minimum distance of 100 to exclude keypoints that are measuring the same feature in the image and limit the complexity of our first-order estimation process in the presence of multi-modal distance distributions. Reducing the number of keypoints also helps with speed and complexity of the O(n choose 2) operation. We then compute the ratio of the relative distances between keypoint matches, and the resulting TTC estimate between frames in the video sequence.

Some examples are included below (for a complete table of results, see the last section of this report):

Example 1: AKAZE Detector, AKAZE Descriptor

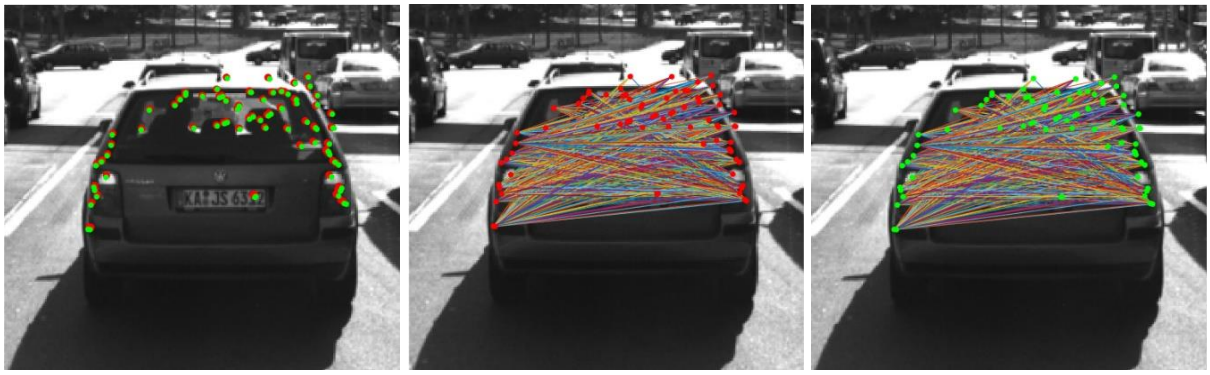


Figure 33: Relative Distances Between AKAZE detectors and AKAZE descriptors on Frames 17 and 18

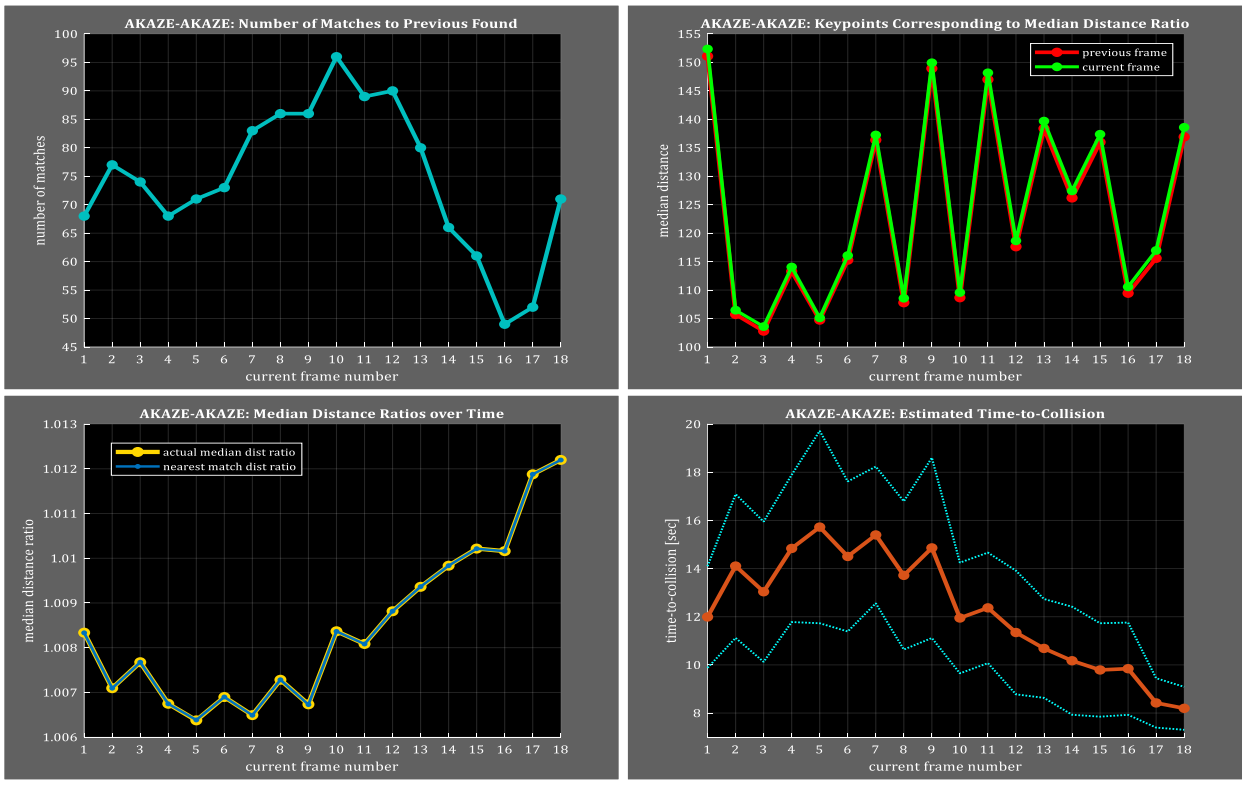


Figure 34: Median Distance Ratio and Time-to-Collision across all frames for AKAZE Detectors & AKAZE Descriptors

Example 2: BRISK Detector, BRIEF Descriptor

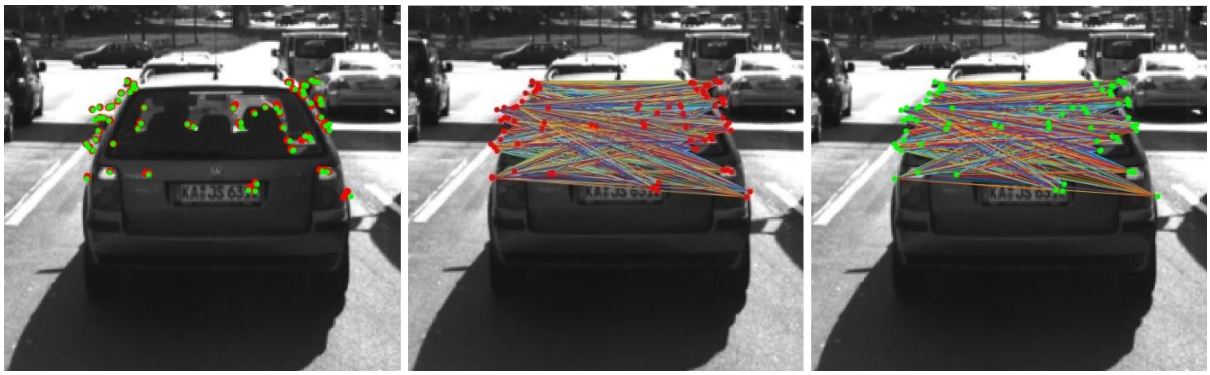


Figure 35: Relative Distances Between BRISK Detectors and BRIEF Descriptors Between Frames 14 and 15

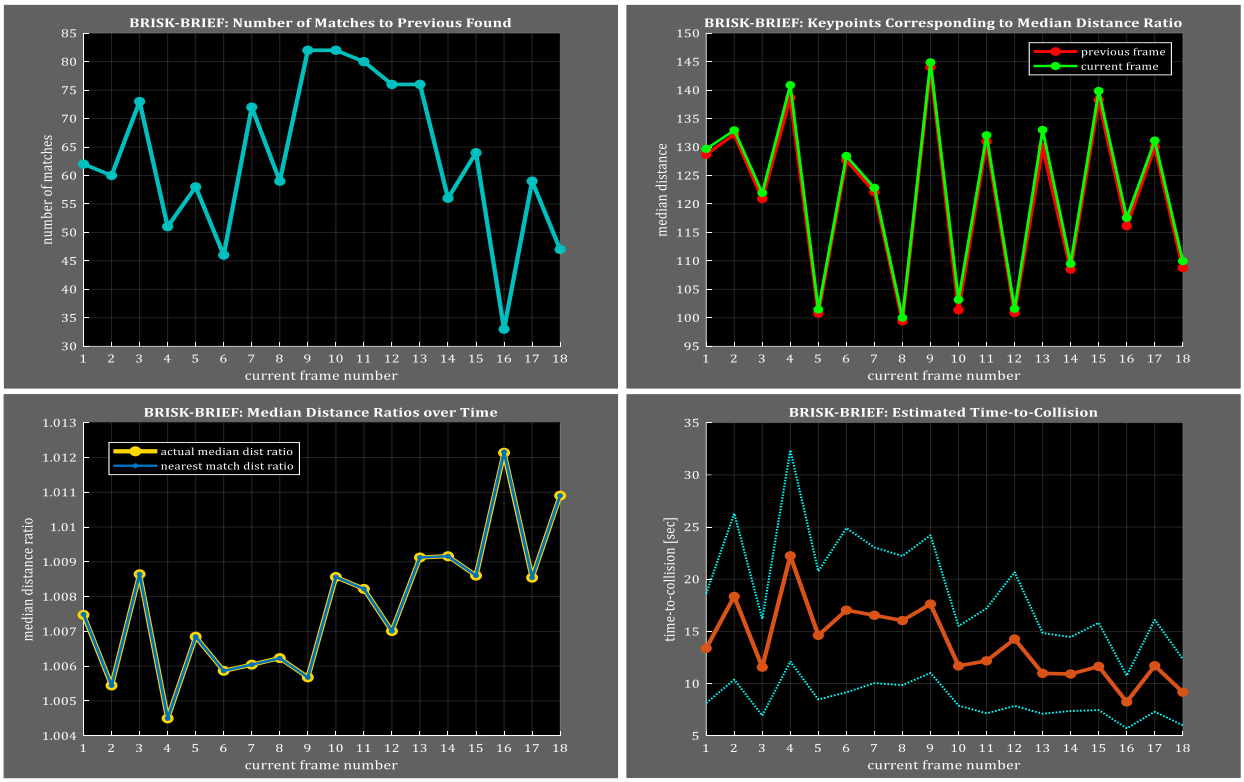


Figure 36: Median Distance Ratio and Time-to-Collision across all frames for BRISK Detectors & BRIEF Descriptors

Example 3: FAST Detector, BRISK Descriptor

Figure 37: Relative Distances Between FAST Detectors and BRISK Descriptors Between Frames 9 and 10

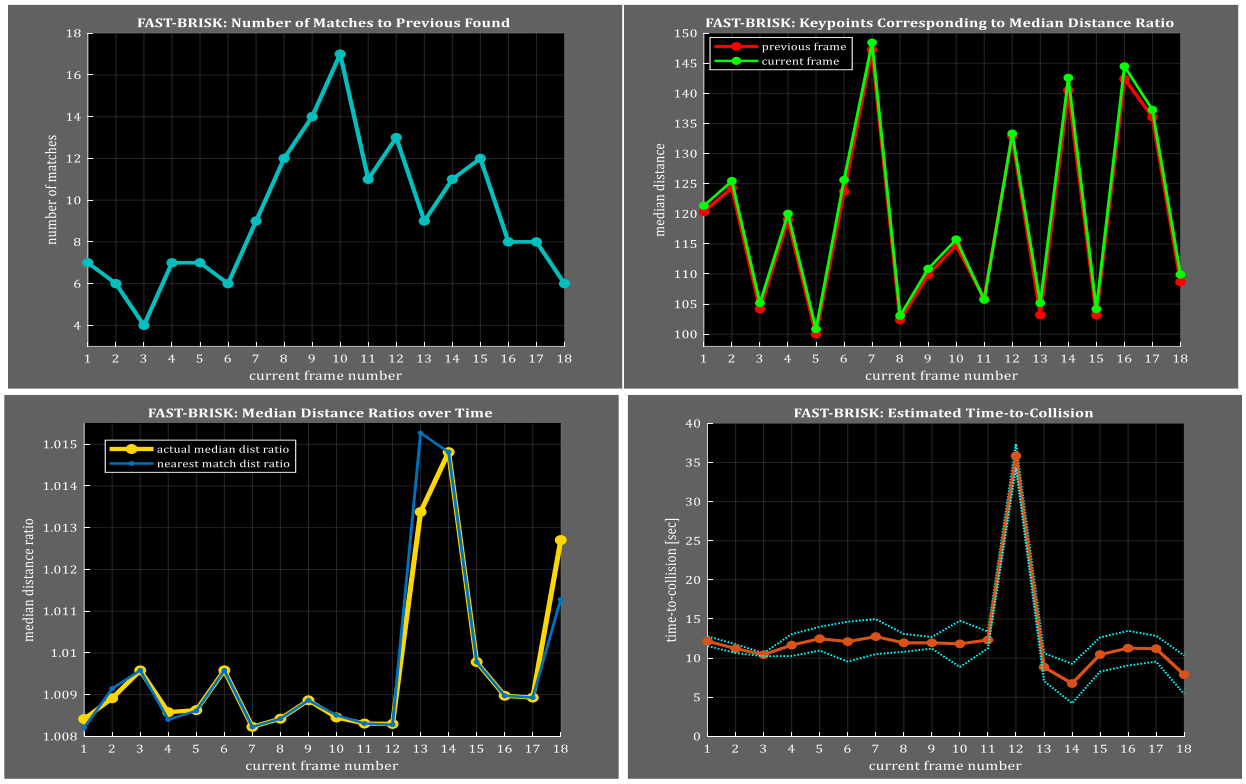


Figure 38: Median Distance Ratio and Time-to-Collision across all frames for FAST Detectors & BRISK Descriptors

Example 4: HARRIS Detector, FREAK Descriptor

Figure 39: Relative Distances Between HARRIS Detectors and FREAK Descriptors Between Frames 6 and 7

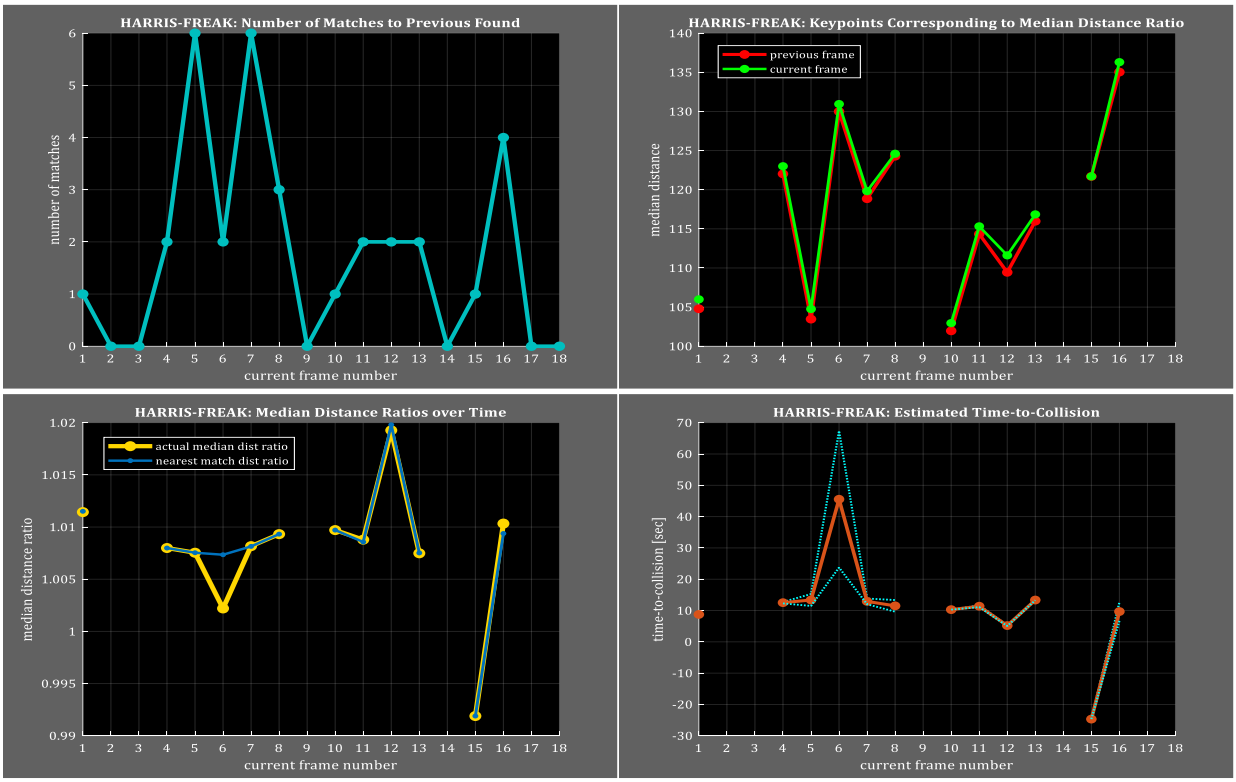


Figure 40: Median Distance Ratio & Time-to-Collision across all frames for HARRIS Detectors & FREAK Descriptors

Example 5: ORB Detector, ORB Descriptor

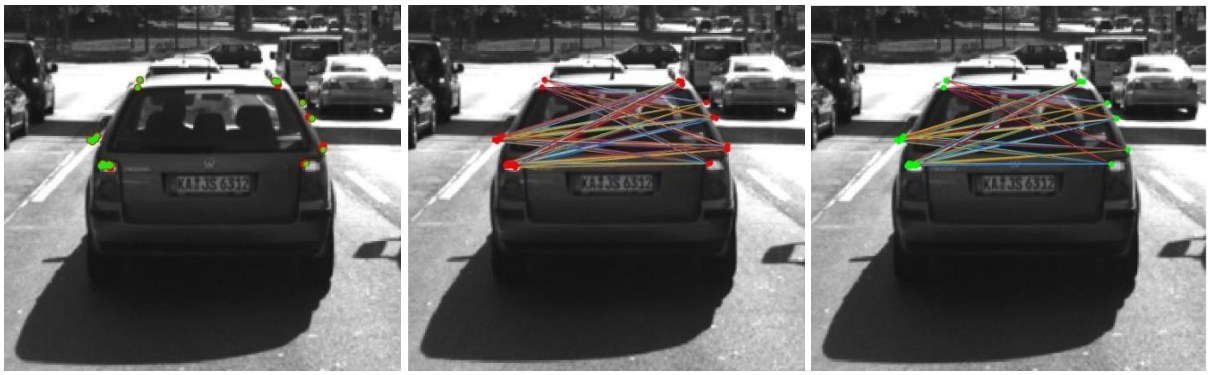


Figure 41: Relative Distances Between ORB Detectors and ORB Descriptors Between Frames 6 and 7

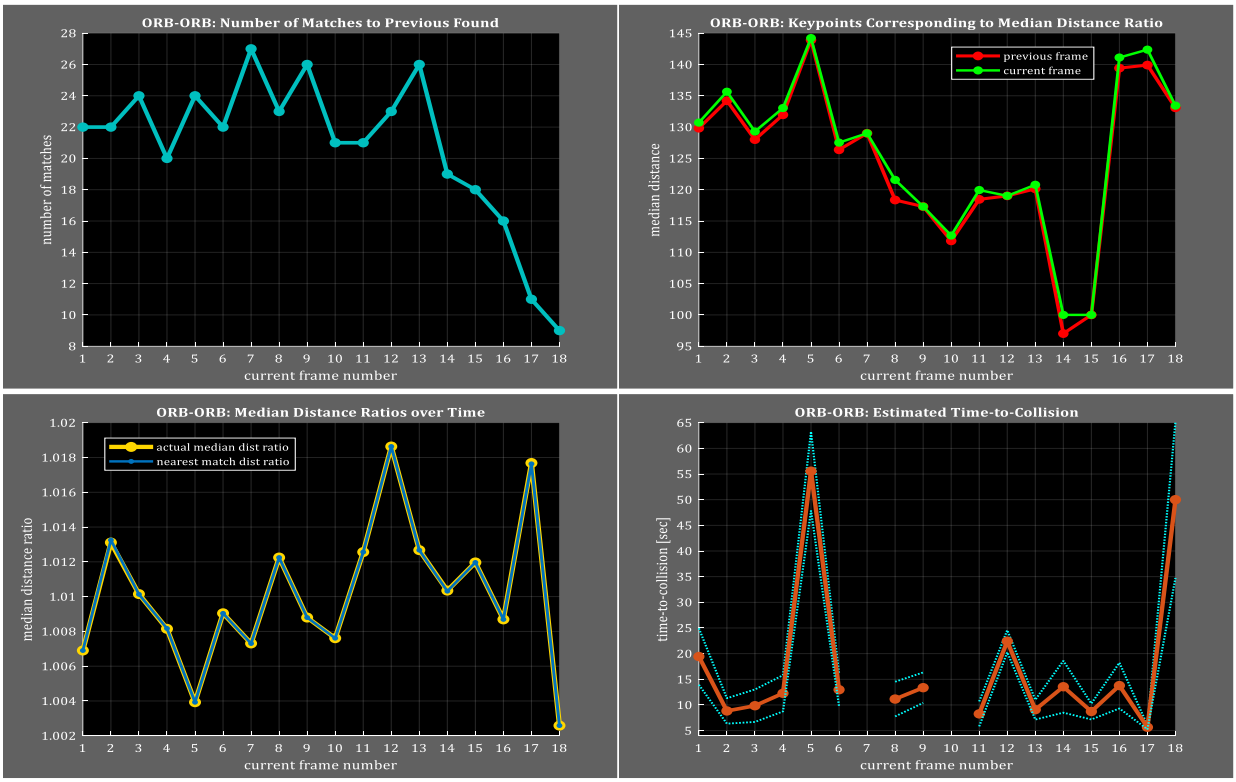


Figure 42: Median Distance Ratio and Time-to-Collision across all frames for ORB Detectors & ORB Descriptors

Example 6: SIFT Detector, SIFT Descriptor

Figure 43: Relative Distances Between SIFT Detectors and SIFT Descriptors Between Frames 14 and 15

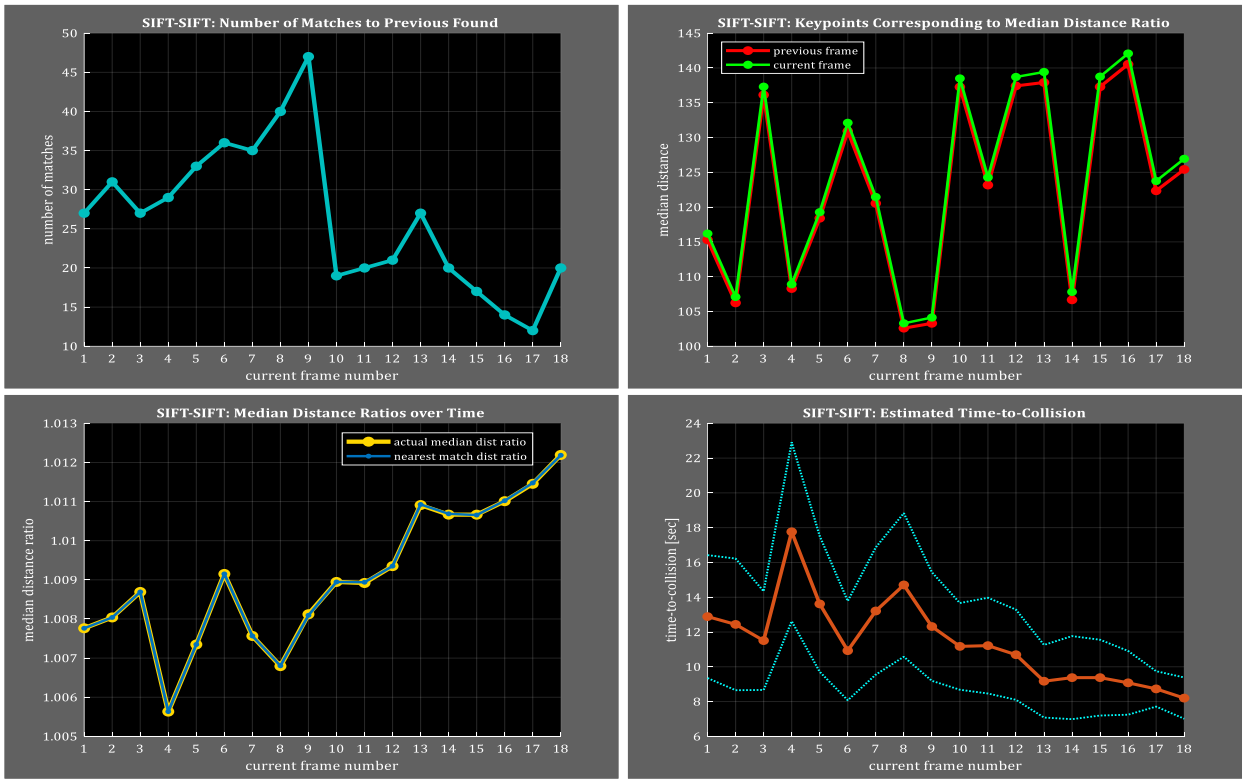


Figure 44: Median Distance Ratio and Time-to-Collision across all frames for SIFT Detectors & SIFT Descriptors

Example 7: Shi-Tomasi Detector, BRIEF Descriptor

Figure 45: Relative Distances Between Shi-Tomasi Detectors and BRIEF Descriptors Between Frames 11 and 12

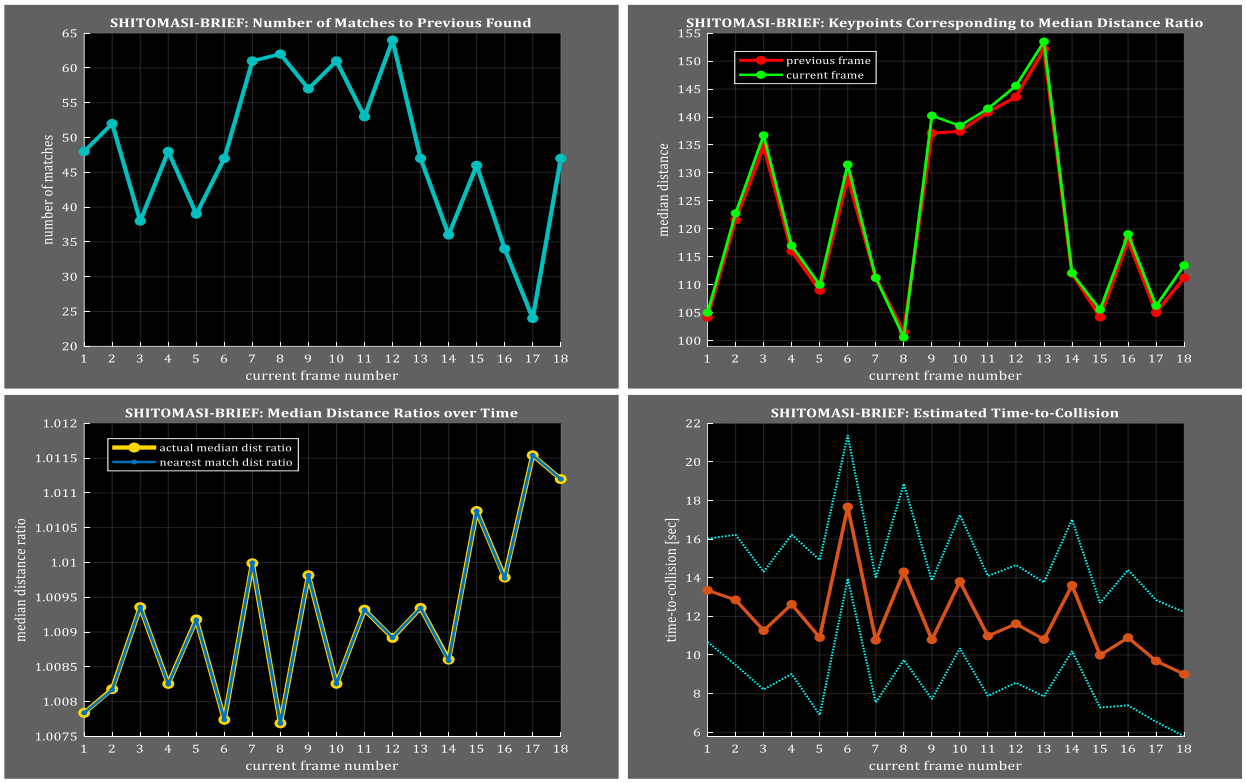


Figure 46: Median Dist Ratio & Time-to-Collision across all frames for Shi-Tomasi Detectors & BRIEF Descriptors

Function Definition for Estimating the Camera-Based TTC in camFusion_Student.cpp:

```
// Compute time-to-collision (TTC) based on keypoint correspondences in successive images
void computeTTCCamera(std::vector<cv::KeyPoint> &kptsPrev, std::vector<cv::KeyPoint> &kptsCurr,
                      std::vector<cv::DMatch> kptMatches, double frameRate, double &TTC, cv::Mat *visImg)
{
    // compute distance ratios for all matched keypoints between the current and previous frame
    vector<double> distRatios;

    for (auto it1 = kptMatches.begin(); it1 != kptMatches.end() - 1; ++it1)
    {
        // get current keypoint and its matched partner in the previous frame
        cv::KeyPoint kpOuterCurr = kptsCurr.at(it1->trainIdx);
        cv::KeyPoint kpOuterPrev = kptsPrev.at(it1->queryIdx);

        for (auto it2 = kptMatches.begin() + 1; it2 != kptMatches.end(); ++it2)
        {
            double minDist = 100.0; // min required distance

            // get next keypoint and its matched partner in the previous frame
            cv::KeyPoint kpInnerCurr = kptsCurr.at(it2->trainIdx);
            cv::KeyPoint kpInnerPrev = kptsPrev.at(it2->queryIdx);

            // compute distances and distance ratios
            double distCurr = cv::norm(kpOuterCurr.pt - kpInnerCurr.pt);
            double distPrev = cv::norm(kpOuterPrev.pt - kpInnerPrev.pt);

            // prevent divide by zero
            if (distPrev > std::numeric_limits<double>::epsilon() && distCurr >= minDist)
            {
                double distRatio = distCurr / distPrev;
                distRatios.push_back(distRatio);
            }
        } // end inner loop over all matched keypoints
    } // end outer loop over all matched keypoints
```

```
    // only continue if list of distance ratios is not empty
    if (distRatios.size() == 0)
    {
        TTC = NAN;
        return;
    }

    // compute median distance ratio, removing outlier influence
    std::sort(distRatios.begin(), distRatios.end());

    long medianIdx = floor(distRatios.size() / 2.0);
    double medianDistRatio;
    if (distRatios.size() % 2 != 0) // if the number of distance ratio elements is odd
    {
        // then take the middle value
        medianDistRatio = distRatios[medianIdx];
    }
    else // if the number of distance ratio elements is even
    {
        // then take the average of the two middle values
        medianDistRatio = (distRatios[medianIdx - 1] + distRatios[medianIdx]) / 2;
    }

    double dT = 1 / frameRate;
    TTC = -dT / (1 - medianDistRatio);
}
```

Function Call in main():

```
    computeTTCCamera((dataBuffer.end() - 2)->keypoints,
                     (dataBuffer.end() - 1)->keypoints,
                     currBB->kptMatches,
                     sensorFrameRate,
                     ttcCamera,
                     imgIndex,
                     kptTTCfilename);
```

FP.5 PERFORMANCE EVALUATION, PART I

> *Find examples where the TTC estimate of the Lidar sensor does not seem plausible. Describe your observations and provide a sound argument as to why you think this happened.*

The figure below shows the lidar TTC and relative velocity estimates. The mint green line is the result from the code implementation. The dark orange line is the TTC based my “eyeball” estimate of the distances to the preceding vehicle. The assertion of where and when the TTC is off is based upon comparisons between the implementation results and manual “eyeball” estimates of the distance to the rear of the preceding vehicle from a top-down view of the lidar points.

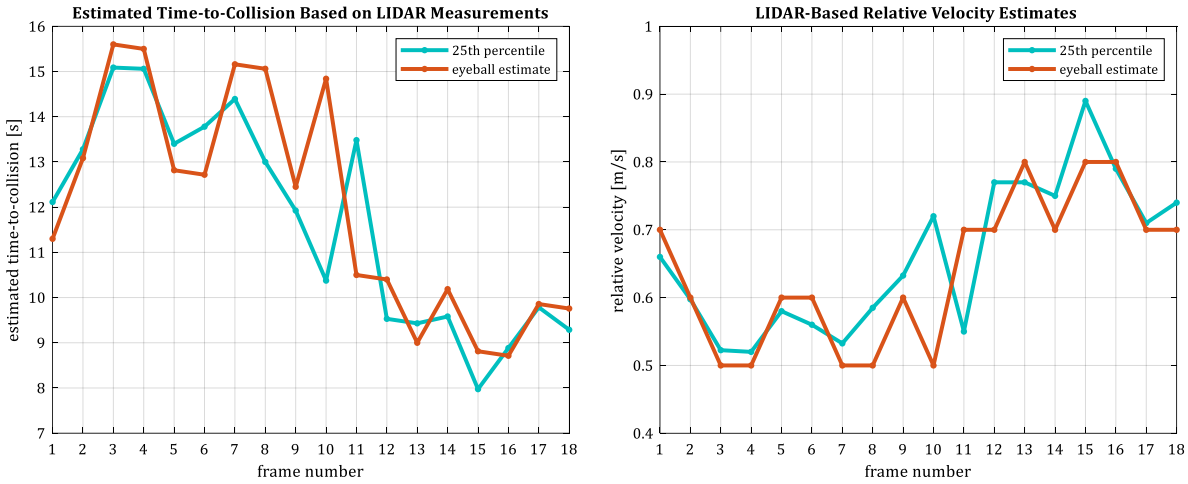
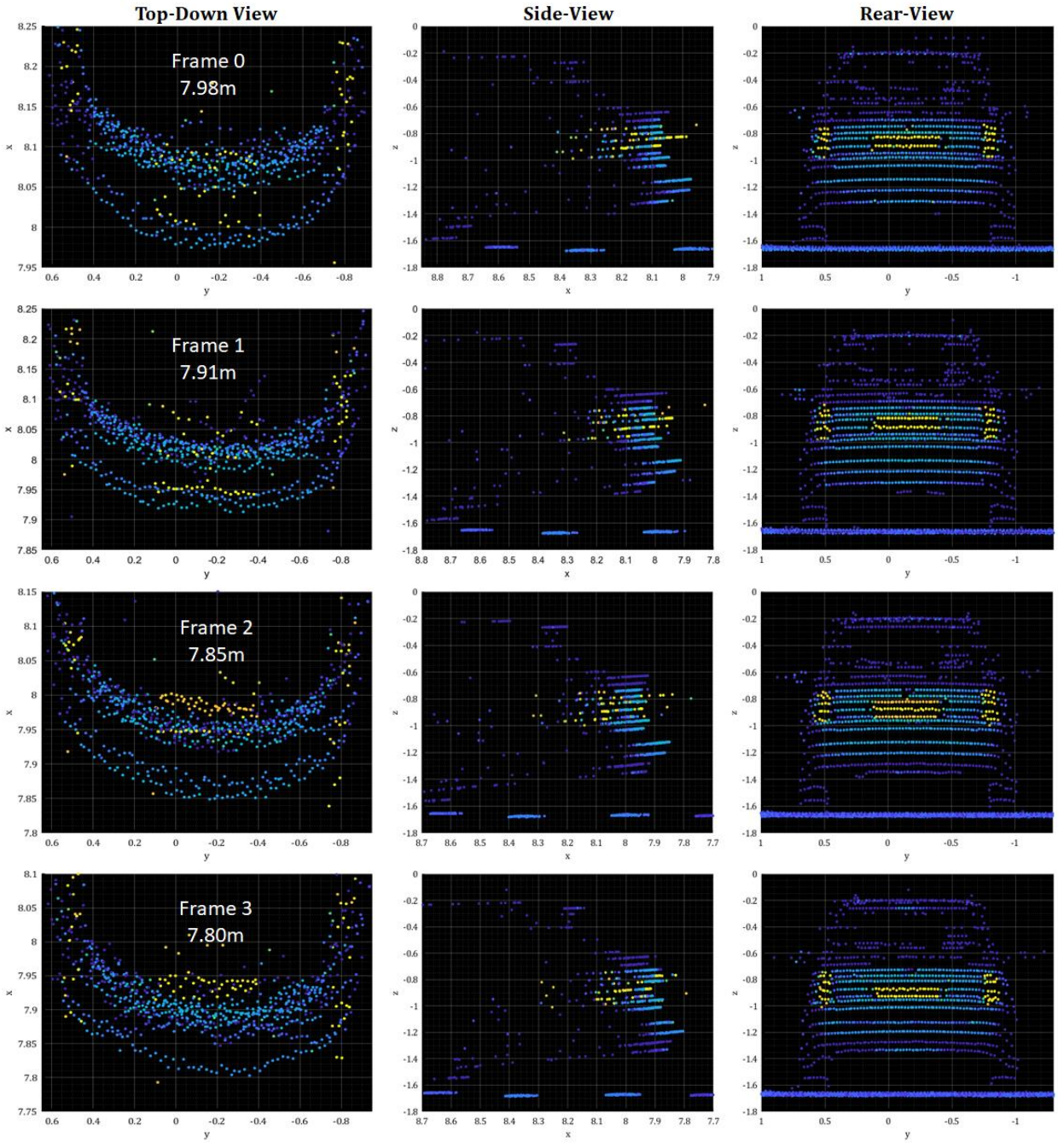
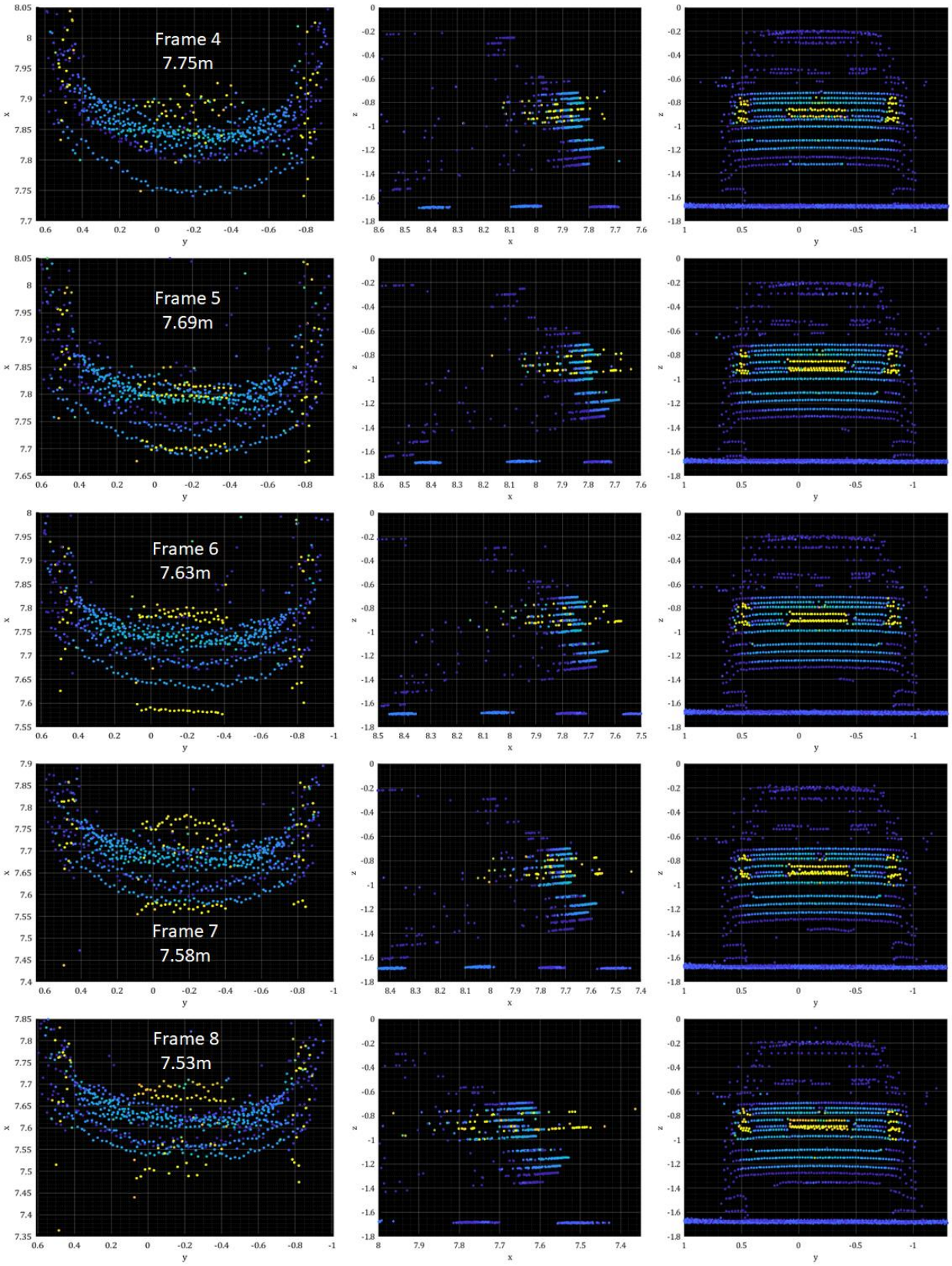


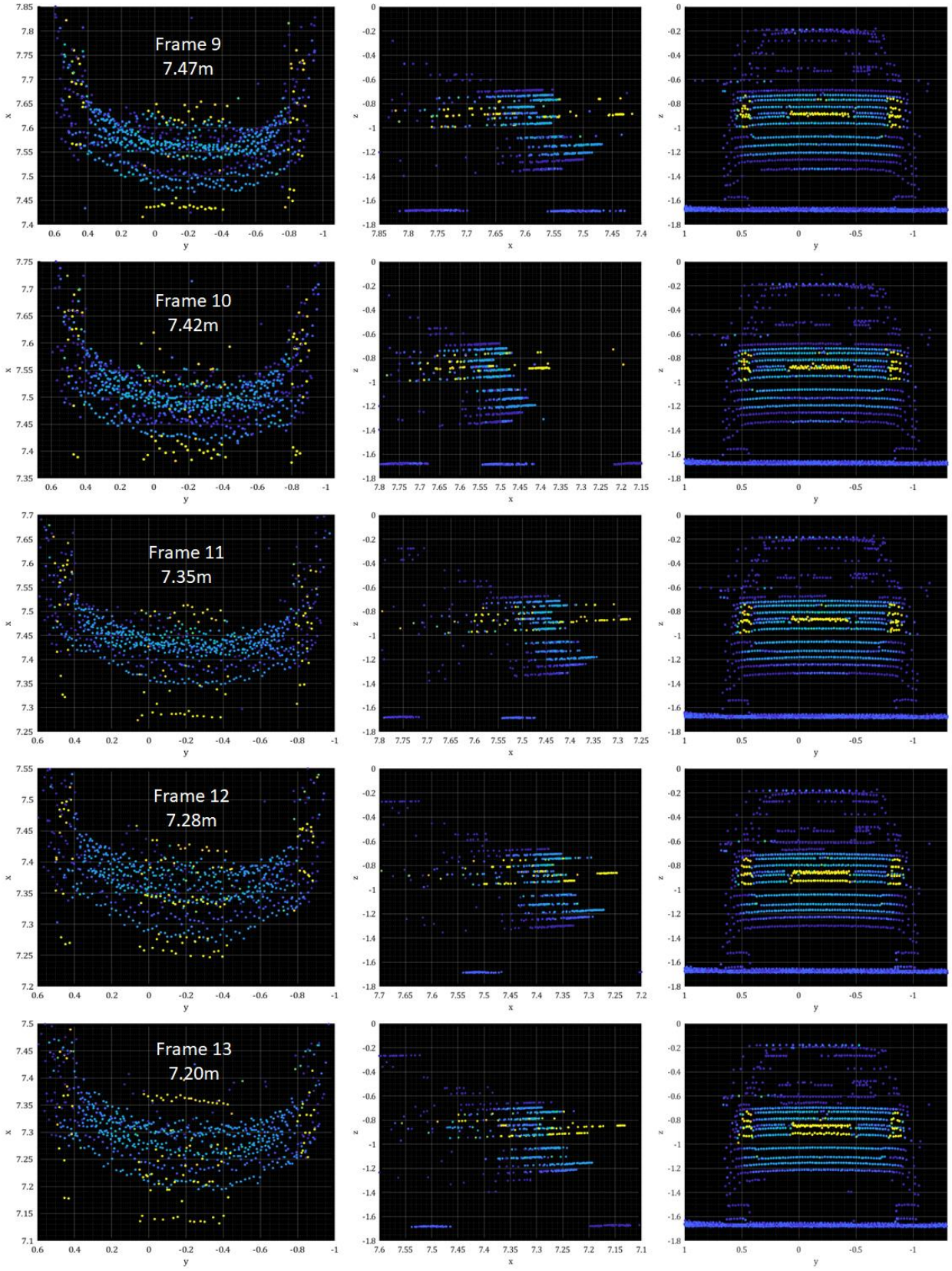
Figure 47: Distribution of LIDAR Time-to-Collision Estimates Based on x-Position Data on the Preceding Car

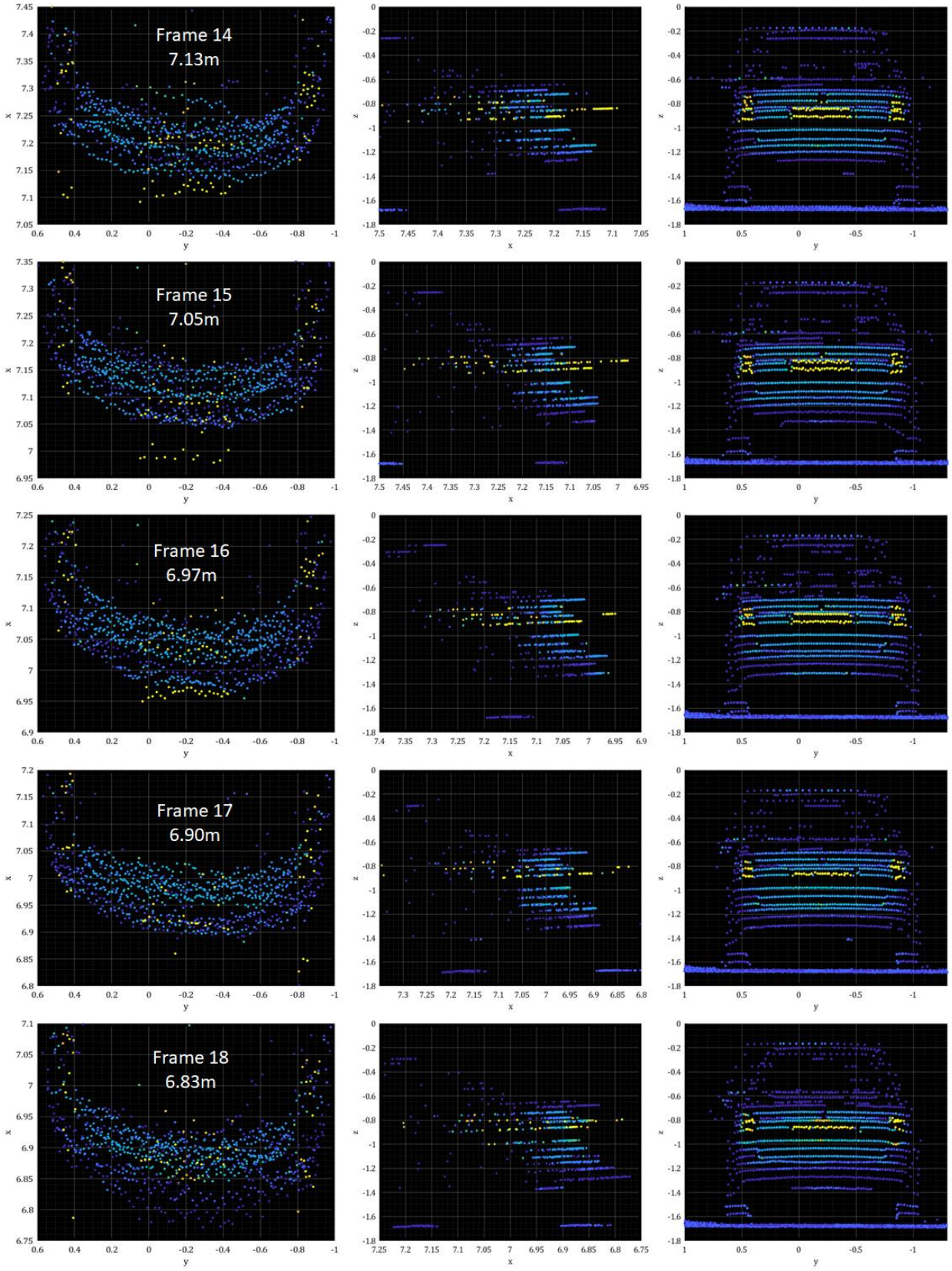
Selection of the lower 25th percentile distance of all lidar points within the region of interest produced relatively stable and plausible estimates of TTC over time. However, the 25th percentile is not the usually the closest point, thus producing estimates of TTC that are biased high relative to truth. This section will explore the errors in the lower percentiles.

I performed the “eyeball” estimates by looking at the top-down views, using some “human judgement” and experience to discern where the closest part of the vehicle might be. This approach was taken due to the absence of a truth estimate to help assess the plausibility of the algorithm results. The x-distance estimates that I selected are shown in the top-down scatter plot for each frame shown below.




Frame 4
7.75m
Frame 5
7.69m
Frame 6
7.63m
Frame 7
7.58m
Frame 8
7.53m


Frame 9
7.47m
Frame 10
7.42m
Frame 11
7.35m
Frame 12
7.28m
Frame 13
7.20m

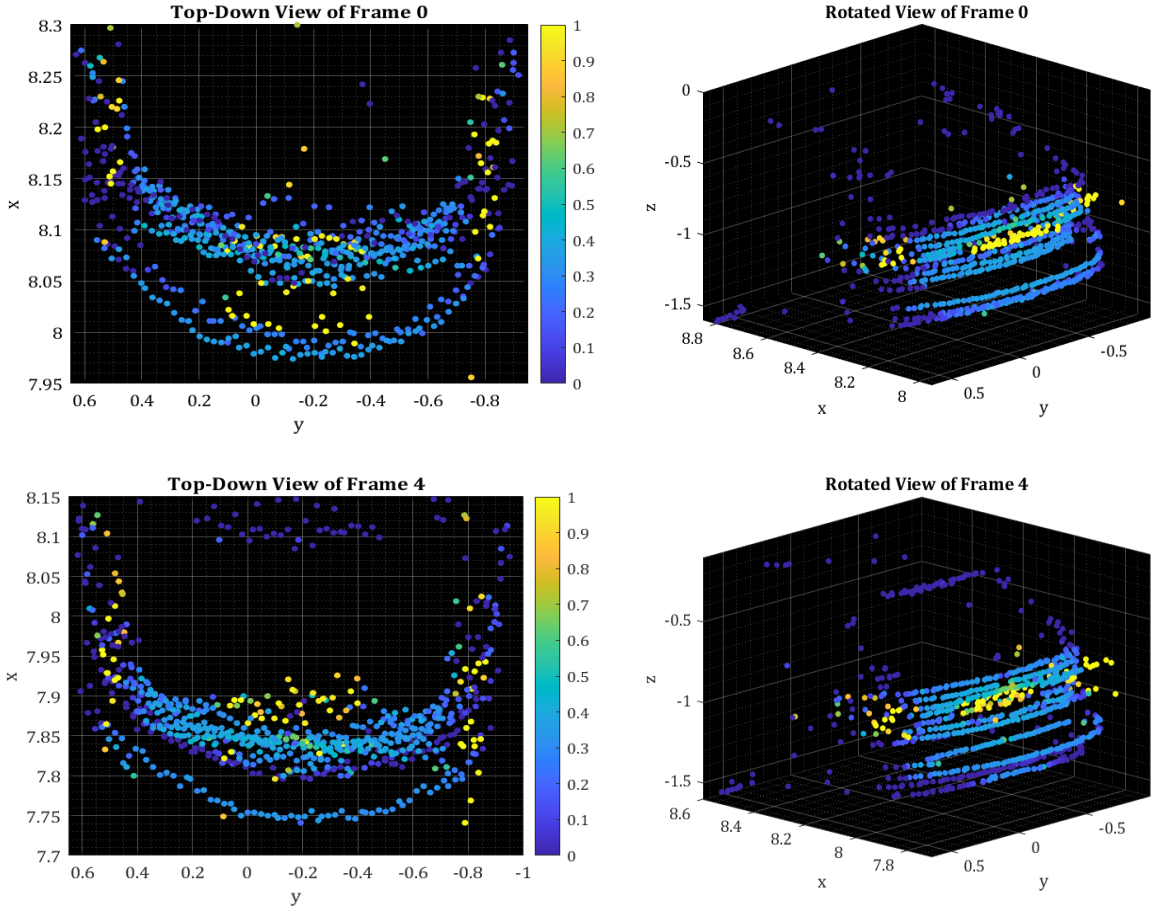

Frame 14
7.13m
Frame 15
7.05m
Frame 16
6.97m
Frame 17
6.90m
Frame 18
6.83m

The top-down (y-x) and side (x-z) views of the preceding vehicle show that the lidar estimates in the x-direction tends be very noisy. By contrast, lidar estimates from the rear (y-z) view of the vehicle reveals a high level of detail, precision and accuracy. I used the rear (y-z) views as a color-coded reference to identify the corresponding point position estimates in the x-direction.

Lidar measurement accuracy is correlated to the amount of light reflected from an object. The lecture notes suggested removing measurements with low reflectivity, explaining that they would be less reliable than points with higher reflectivity. However, I found the opposite to be true ...

The scatter plots showed distinctively higher reflectivity estimates coming off the vehicle license plate and tail-lights. These higher reflectivity points demonstrated some interesting behaviors – sometimes they were noisier than less reflective points (especially on the tail-lights), other times they organized into distinct scan lines (i.e. on the license plate) with conflicting information as to where these surfaces truly existed relative to the rest of the vehicle in the x-direction.

The following images show the lidar measurements on the preceding car (selected bounding box) across different perspectives: (1) a top-down view in world coordinates, and (2) 4D rotated view in world coordinates. In addition to position data, the reflectivity estimates of the measurements have been color-coded according to the scale shown, with 0 being the lowest reflectivity and 1 being the highest reflectivity observed across all lidar measurements at that snapshot in time.



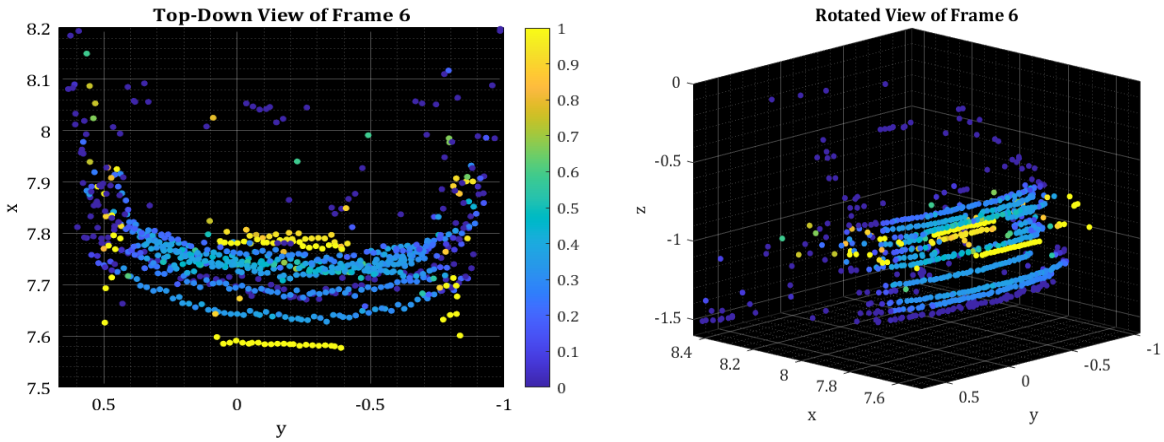


Figure 48: Comparison of Lidar Point Cloud Distributions on High Reflectivity Features

For the tail-lights, the distance measurements in the x-direction are particularly high. This could be due a combination of the surface materials and the shape curvature at the edges of the vehicle, which would affect the directions of scattered light and the intensity of the return signal. Lower reflectivity points along the edge do not show the same magnitude of error, and the noise on the similarly reflective license plate are high on some (but not all) frames.

Relative motion would cause the aspect angle of the lidar sensor to change across frames. It also changes at a much smaller scale within a frame as a function of the elevation of each individual lidar scan. The slight variations in aspect angle and corresponding light scatters at different elevations could explain why some parts of the license plate appear at 7.6m and other parts at 7.8m in Frame 6 shown above. The azimuth off boresight could also explain why the noise is higher on the tail-lights.

The white license plate contains black painted letters and numbers that may contribute to small angle-dependent variations across the surface of the plate. The lidar sensor may also be picking up variations in ambient light, shadows and reflections from passing vehicles. Car exhaust particles and other environmental factors may also contribute to measurement noise as the vehicles gather at the stop light.

Measurement noise in the x-direction is especially problematic for a TTC computation that is highly-sensitive to small errors in absolute and relative distance across frames. Recall the equation we used: $TTC = (d_1 * \Delta t)/(d_0 - d_1)$. Smaller denominators produce TTC estimates that approach positive or negative infinity very quickly. Even one centimeter of error in the distance measurement on any frame in our sequence could produce dramatic errors in the estimated TTC.

The TTC equation also makes a simplifying assumption of constant velocity for small Δt. Δt in our case is 1/10th of a second. That sampling rate would be “small” if we were cruising on an uncrowded highway at a higher speed. However, the vehicles in our scenes are approaching a stop light, where the relative velocity between the ego car and the preceding car is not constant -- we can even observe the occasional red flicker of the brake lights as the driver in the preceding vehicle places intermittent pressure on the brake. The figure below shows the relative velocity based on percentiles across the distribution of x-distances in the region of interest. We can see that the magnitude of the fluctuations decrease as the percentiles increase. However, it is still unclear how much of the fluctuations across frames are due to noise versus actual changes in velocity.

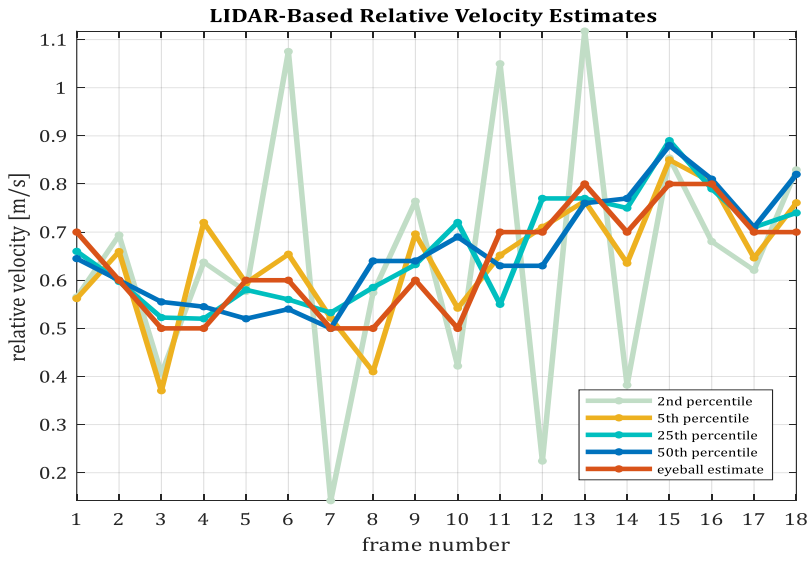


Figure 49: Change in x-distance measurements as a function of time between adjacent frames

In summary, lidar x-distance measurements can be noisy. Errors in the TTC estimates based on the constant velocity model are expected to be highly-sensitive to measurement noise and fluctuations in velocity that occur below the sampling rate. Furthermore, the constant-velocity assumption does not generalize well to dynamic traffic situations where vehicles are braking hard, or where we don't have a centered reference point to compare to over time.

For real-life applications, it is recommended that we use the constant-acceleration model for computing the lidar-based time-to-collision.

FP.6 PERFORMANCE EVALUATION, PART II

Run several detector-descriptor combinations and evaluate the differences in TTC estimation. Identify which methods perform the best. Find examples where camera based TTC estimation is way off, describe observations and investigate potential reasons.

Several detector-descriptor combinations were compared with respect to the TTC estimate on a frame-by-frame basis. The results are shown in the tables below:

Median Camera-Based TTC Estimate

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
AKAZE AKAZE	11.99	14.1	13.04	14.84	15.72	14.5	15.4	13.72	14.86	11.95	12.37	11.34	10.68	10.17	9.79	9.843	8.421	8.197
AKAZE BRIEF	13.51	15.76	13.79	13.76	15.08	14.37	15.97	14.18	14.6	12.25	12.22	11.56	10.08	9.403	9.259	9.847	9.074	8.197
AKAZE BRISK	12.02	14.33	12.97	13.85	14.88	13.81	15.5	13.72	14.2	11.91	12.59	11.59	10.05	9.606	9.662	9.828	9.153	8.333
AKAZE FREAK	12.92	13.85	13.66	13.81	14.36	13.83	15.6	13.44	14.59	12.2	12.55	11.06	10.22	9.737	9.935	9.814	9.017	8.217
AKAZE ORB	12.79	14.4	13	14.75	15.36	14.47	15.87	14.06	14.16	12.22	12.59	11.66	10.01	9.718	10.06	9.47	9.179	8.45
AKAZE SIFT	12.25	14.38	13.11	14.75	16.26	14.77	15.72	13.95	14.76	12	12.55	11.42	10.8	9.921	9.368	10.09	9.017	8.347
BRISK BRIEF	13.38	18.35	11.57	22.25	14.62	17.04	16.54	16.04	17.62	11.7	12.18	14.27	10.98	10.92	11.64	8.241	11.71	9.174
BRISK BRISK	12.22	18.94	11.93	15.34	28.9	14.2	18.98	15.05	13.96	11.72	13.35	11.39	10.96	10.09	11.3	9.238	9.268	9.747
BRISK FREAK	12.82	24.33	10.04	14.35	31.25	15.86	16.13	16.16	17.95	10.93	13.77	11.17	10.89	11.12	13.25	7.924	9.852	8.961
BRISK ORB	12.76	17.7	11.25	15.79	17.42	17.04	16.16	18.26	15.12	10.68	10.71	11.47	11.26	12.92	13.05	9.501	9.158	10.4
BRISK SIFT	14.09	17.67	12.5	15.34	29.67	18.62	15.85	15.07	17.79	11.56	14.33	11.83	10.32	10.16	11.68	8.446	10.31	9.183
FAST BRIEF	11.1	13.79	10.44	12.57	11.58	22.47	59.17	10.38	11.83	12.67	13.04	13.26	9.372	10.93	11.48	11.29	11.17	
FAST BRISK	12.17	11.21	10.44	11.66	12.48	12.11	12.75	11.95	11.95	11.82	12.32	35.84	8.85	6.752	10.46	11.27	11.2	7.874
FAST FREAK	11.36	13.08	10.44	11.66	11.45	11.03	40.98	10.29	12.11	11.82	12	12.3	9.872	6.752	10.33	11.68	11.8	7.874
FAST ORB	11.1	13.79	10.44	12.57	11.58	-Inf	59.17	10.38	11.83	11.2	13.04	13.26	9.385	9.264	10.16	11.29	11.17	
FAST SIFT	11.14	13.79	10.44	12.57	11.58	12.11	59.17	10.34	11.83	11.82	13.12	12.66	8.881	9.264	8.503	11.29	11.8	9.921
HARRIS BRIEF	8.753	-17.2		12.52	13.08	-9.503	15.82	18.9		-6.72	11.38	5.189	26.81		-14.78	11.77		
HARRIS BRISK	8.753			12.52	19.69		11.95	18.9			11.38	5.189	26.81		-14.78	10.65		
HARRIS FREAK	8.753			12.52	13.37	45.55	12.92	11.49		10.29	11.38	5.189	13.35		-24.68	9.676		
HARRIS ORB	8.753	-17.2		12.52	13.08	27.25	13.61	18.9			11.38	5.189	26.81	3.449	-14.78	10.65		
HARRIS SIFT	8.753	-17.2		12.52	13.08	-33.91	13.61	18.9		-6.72	11.38	5.189	26.81	-38.88	-14.78	10.11		
ORB BRIEF	15.84	132.5	26.53	10.73	25.19	16.34	10.22	27.1	43.86	97.09	14.08	12.86	12.85	7.657		18.2	7.596	21.81
ORB BRISK	16.5	9.592	9.533	14.06	500	7.018	96.62	8.432	13.12	11.06	7.452	47.62	7.452	25.06	8.658	12.12	8.857	57.64
ORB FREAK	11.51	10.76	10.55	10.87	75.76	9.276	-Inf		12.57	-Inf	7.955	38.31	6.325	54.2				
ORB ORB	19.44	8.834	9.852	12.27	55.56	12.97	-Inf	11.15	13.37	-Inf	8.261	22.42	9.083	13.55	8.726	13.77	5.655	50
ORB SIFT	19.42	9.39	8.34	11.55	526.3	12.97	-Inf	10.86	13.76	-Inf	8.838	28.74	7.539	12.92	21.98	13.28	6.059	48.08
SHITOMASI BRIEF	13.35	12.85	11.26	12.63	10.91	17.67	10.76	14.31	10.79	13.8	10.98	11.61	10.8	13.61	9.99	10.9	9.69	9.005
SHITOMASI BRISK	13.37	11.91	12.11	12.74	12.48	13.32	12.02	13.89	11.53	12.71	10.08	11.67	11.01	11.03	9.124	10.79	10.98	8.368
SHITOMASI FREAK	13.37	13.03	12.03	12.67	12	14.77	11.92	12.67	11.42	13.02	10.09	12.03	11.59	11.25	9.208	10.35	9.013	7.855
SHITOMASI ORB	13.18	12.67	11.73	11.67	11.86	13.93	9.804	13.93	10.86	13.32	10.67	11.72	11.01	11.66	9.901	10.46	11.15	7.731
SHITOMASI SIFT	13.18	12.53	11.04	12.38	12.03	14.62	9.804	14.29	11.53	13.41	10.85	11.64	10.96	10.48	11.25	6.859	10.21	7.752
SIFT BRIEF	13.56	11.83	10.85	17.78	14.93	12.05	13.59	14.53	12.72	11.47	12.77	10.5	9.643	8.726	9.843	9.276	8.977	8.268
SIFT BRISK	12.4	13.09	11.51	19.05	14.22	11.76	14.5	15.28	13.05	11.41	12.17	10.47	9.434	9.671	9.368	8.734	9.355	8.45
SIFT FREAK	12.36	13.09	11.66	18.18	15.38	12.97	13.26	15.82	12.59	10.48	12.3	10.46	9.149	9.452	8.822	8.518	9.124	8.46
SIFT SIFT	12.89	12.44	11.51	17.76	13.61	10.93	13.21	14.71	12.32	11.17	11.22	10.7	9.17	9.376	9.376	9.079	8.73	8.2

frame number

Standard Deviation Across Camera-Based TTC Estimates

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
AKAZE AKAZE	72.77	316.8	136.2	113.4	871.4	245.2	35.47	58.57	132	29.77	556.7	301	32.79	91.07	43.46	20.9	10.48	1.833
AKAZE BRIEF	610	342.9	253.3	1812	887.8	105.3	48.44	74.94	128.1	37.07	419.4	101.2	53.47	85.17	28.02	11.78	9.88	2.965
AKAZE BRISK	205.9	347.7	97.92	40.67	882.8	233.1	39.82	42.66	28.01	17.69	674.5	99.06	6.281	24.68	43.9	11.99	9.727	18.42
AKAZE FREAK	242.5	376.9	244.6	36.12	810.4	252.9	49.84	47.36	148.5	31.99	612.9	116.8	94.58	82.91	38.94	24.05	9.882	1.856
AKAZE ORB	257.2	338.4	238.2	1953	816.3	291.6	46.66	60.88	136.3	49.61	423.6	114.8	33.13	32.33	45.19	13.62	10.98	1.854
AKAZE SIFT	194	326.2	202.5	50.42	809.2	248	48.92	30.96	130.2	29.5	549.5	110.1	48.94	77.01	40.67	22.85	10.3	2.949
BRISK BRIEF	198.4	243.2	193	499.1	126.4	217.7	318.2	138.1	408.8	1029	97.11	487.7	576	471.7	274.1	57.24	179	73.65
BRISK BRISK	340.3	3176	500.9	225.8	407	459	3388	212.8	105.1	180.2	254.3	459.4	478.2	181.7	222.4	88.76	31.28	156.4
BRISK FREAK	253.7	3476	724.6	230.3	360.9	585.5	168.7	246.7	135.4	140.9	3475	156.6	540.8	175.4	244.4	21.55	29.57	189.1
BRISK ORB	315.2	3563	536.4	58.9	306.2	173	327.6	239.8	120.1	64.45	306.7	753.8	538.4	595.6	441.1	41.1	63.82	67.57
BRISK SIFT	212.9	4146	596.4	53.45	327.8	457.1	3508	274.7	132.6	75.54	87.04	499.1	490.2	580.1	417.8	37.85	54.09	60.09
FAST BRIEF	3.038	76.99	0.7486	9.394	8.588	26.16	655.5	4.776	2.418	8.695	49.47	11.99	4.921	2.879	15.55	7.667	3.075	
FAST BRISK	0.9152	33.69	0.7486	2.689	7.441	28.01	19.59	10.53	2.393	338.5	48.46	19.39	4.479	3.205	6.508	7.64	2.598	5.137
FAST FREAK	0.7706	30.58	0.7486	2.689	11.14	3.682	1170	3.262	1.284	10.45	2.065	9.719	2.726	3.216	6.485	3.342	2.7	5.137
FAST ORB	3.038	68.48	0.7486	9.394	8.588	32.94	655.5	4.776	2.418	329.6	49.47	11.99	4.961	3.131	6.137	7.667	3.075	
FAST SIFT	2.887	68.48	0.7486	9.394	8.588	22.94	655.5	4.795	2.418	293.2	50.36	15.81	5.256	3.131	5.939	7.667	2.7	10.14
HARRIS BRIEF	0.08126	0		1.006	26.83	14.56	19.31	58.67		0	0.4029	0.202	0.07563		25.24	5.873		
HARRIS BRISK	0.08126			1.006	34.56		0.9513	58.67			0.4029	0.202	0.07563		25.24	3.004		
HARRIS FREAK	0.08126			1.006	29.96	26.18	1.063	10.99		0	0.4029	0.202	0.07563		0	2.994		
HARRIS ORB	0.08126	0		1.006	26.83	24.44	17.81	58.67			0.4029	0.202	0.07563	0	25.24	3.004		
HARRIS SIFT	0.08126	1.393		1.006	26.83	18.48	17.81	58.67		0	0.4029	0.202	0.07563	0	25.24	3.135		
ORB BRIEF	130.7	364.4	915.5	200.5	184.2	56.19	1651	74.54	87.27	300.6	43.83	39.73	18.48	39.1		44.6	14.03	227.1
ORB BRISK	140.4	163.1	784.9	228.8	123.2	41.89	153.5	24.16	36.69	5.705	5.378	509.2	5.629	48.99	7.668	89.79	6.124	12.71
ORB FREAK	56.69	266.8	27.72	2.72	90.98	35.1	90.3		3.871	56.84	6.908	24.03	2.981	21.65				
ORB ORB	368.9	175.2	705.7	145	53.6	26.31	1483	33.59	80.12	20.76	18.38	10.17	12.04	8.849	22.06	10.97	1.428	18.41
ORB SIFT	295.6	182.5	10.79	163.8	148.6	23.85	133.8	33.25	60.69	16.6	4.344	16.82	8.285	19.41	16.99	10.73	1.437	18.38
SHITOMASI BRIEF	85.85	119.6	65.91	28.9	9.204	195.7	207.2	252.5	111.2	166.2	101.2	91.71	42.34	69.51	32.64	284.8	190.1	16.21
SHITOMASI BRISK	85.36	74.9	63.46	18.7	130.7	26.3	199.2	240.2	131	126.8	140.3	12.1	17.37	47.84	7.693	313	212.8	12.72
SHITOMASI FREAK	88.01	26.7	55.86	30.09	115.3	43.13	244.5	202.6	146.4	138.3	146.2	14.81	41	25.13	13.59	301.3	202.2	12.55
SHITOMASI ORB	78.22	120.5	63.1	22.59	109.9	161.3	223.4	298	13.19	187.3	121.7	93.28	36.07	79.86	31.17	321.1	179.1	8.001
SHITOMASI SIFT	78.22	123.7	60.81	25.61	111.9	180.7	223.4	252.3	122.4	170.7	128.4	92.15	38.19	42.22	32.22	11.22	174.7	40.16
SIFT BRIEF	595.4	51.72	222.3	79.11	672.6	161.9	22.39	96.88	15.88	6.706	20.76	256.4	205.4	7.891	7.63	368.8	2.825	33.57
SIFT BRISK	601.3	56.38	230.8	88.52	714.1	165.5	20.19	103.8	17.35	6.695	21.42	46.29	214.6	7.77	7.245	294.9	2.234	40.99
SIFT FREAK	757.5	46.14	250.9	87.08	421.8	177.4	22.47	111.6	17.29	7.413	8.628	53.26	230.1	8.69	8.128	13.09	2.303	41.54
SIFT SIFT	478.4	46.66	201.2	192.6	503.4	135.6	23.87	96.45	22.94	6.26	7.216	232.5	4.837	7.325	7.083	240.9	2.045	28.83

frame number

Median Absolute Deviation Across Camera-Based TTC Estimates

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
AKAZE AKAZE	2.111	2.985	2.911	3.057	3.995	3.115	2.838	3.079	3.743	2.299	2.295	2.568	2.052	2.244	1.939	1.916	1.026	0.8921
AKAZE BRIEF	3.507	5.34	3.106	2.688	3.773	3.346	3.455	3.584	4.536	2.344	2.853	2.713	1.999	1.851	1.831	1.74	1.406	1.106
AKAZE BRISK	2.473	3.098	2.434	2.381	3.632	2.797	3.305	3.052	3.756	2.067	2.564	2.55	1.633	1.846	1.53	1.89	1.209	0.9538
AKAZE FREAK	3.009	2.75	3.12	2.739	3.743	2.902	3.432	3.236	3.95	2.438	2.421	2.109	2.167	2.256	1.812	1.877	1.208	0.9845
AKAZE ORB	2.875	3.691	2.649	2.848	4.265	3.065	3.284	3.183	4.003	2.494	2.863	2.553	1.864	2.06	2.003	1.838	1.339	0.9896
AKAZE SIFT	2.271	3.412	3.244	3.31	4.479	3.201	3.194	3.011	3.767	2.307	2.442	2.394	2.095	2.355	2.286	1.977	1.241	0.8953
BRISK BRIEF	5.245	7.962	4.626	10.13	6.15	7.87	6.501	6.191	6.594	3.81	5.029	6.413	3.867	3.542	4.183	2.528	4.405	3.176
BRISK BRISK	4.431	7.503	4.549	6.415	16.3	5.928	8.006	5.943	5.371	4.045	4.884	4.556	3.721	3.094	4.382	2.643	3.362	3.483
BRISK FREAK	4.294	13.74	3.843	5.26	17	6.727	6.71	6.18	6.737	3.527	4.801	4.092	3.433	2.815	4.935	2.208	3.337	3.028
BRISK ORB	4.521	6.937	4.52	6.482	7.204	6.258	6.848	7.885	5.251	3.876	3.955	4.627	3.627	3.646	5.106	2.731	3.431	3.505
BRISK SIFT	5.313	6.684	5.527	5.835	15.68	7.438	6.063	5.413	6.7	4.256	5.103	4.751	3.527	3.201	4.992	2.316	4.018	3.24
FAST BRIEF	1.314	2.592	0.1999	1.622	2.049	2.549	22.49	3.626	0.779	2.634	1.154	1.606	2.434	1.735	4.671	2.215	2.149	
FAST BRISK	0.6202	0.5569	0.1999	1.4	1.518	2.549	2.234	1.144	0.7382	2.956	1.071	1.538	1.788	2.513	2.182	2.215	1.647	2.466
FAST FREAK	0.7245	1.761	0.1999	1.4	1.375	2.748	16.88	2.85	0.7962	1.85	0.7367	0.7199	2.294	2.513	2.901	2.573	1.613	2.466
FAST ORB	1.314	2.164	0.1999	1.622	2.049	4.3	22.49	3.626	0.779	2.23	1.154	1.606	2.531	2.648	2.704	2.215	2.149	
FAST SIFT	1.254	2.164	0.1999	1.622	2.049	3.457	22.49	2.716	0.779	2.725	1.211	1.784	2.741	2.648	2.584	2.215	1.613	3.072
HARRIS BRIEF	0.05746	0		0.2395	3.352	0.3292	1.205	2.19		0	0.2849	0.1428	0.05348		1.738	3.692		
HARRIS BRISK	0.05746			0.2395	0.5161		0.6525	2.19			0.2849	0.1428	0.05348		1.738	2.237		
HARRIS FREAK	0.05746			0.2395	1.898	21.82	0.9167	1.856		0	0.2849	0.1428	0.05348		0	2.971		
HARRIS ORB	0.05746	0		0.2395	3.352	2.55	1.21	2.19			0.2849	0.1428	0.05348	0	1.738	2.237		
HARRIS SIFT	0.05746	1.317		0.2395	3.352	17.47	1.21	2.19		0	0.2849	0.1428	0.05348	0	1.738	2.682		
ORB BRIEF	2.242	17.27	13.32	3.513	12.02	4.964	2.553	2.093	5.135	22.42	6.629	4.195	3.806	2.305		8.119	3.385	6.55
ORB BRISK	7.928	2.952	2.852	5.252	19.94	3.269	4.812	1.028	4.683	3.565	2.058	15.59	2.892	5.645	1.402	3.455	3.357	5.276
ORB FREAK	3.035	3.191	1.54	1.266	20.72	2.521	16.16		1.391	15.95	1.487	18.32	2.692	22.72				
ORB ORB	5.5	2.477	3.163	3.531	7.7	3.197	4.71	3.38	2.962	4.852	2.431	2.154	1.924	5.051	1.552	4.455	0.3924	15.25
ORB SIFT	7.035	2.53	2.322	4.431	19.67	3.709	5.067	3.436	3.655	12.59	1.819	5.631	1.862	4.219	1.672	4.73	0.5255	14.7
SHITOMASI BRIEF	2.672	3.372	3.047	3.608	4.016	3.708	3.223	4.56	3.073	3.454	3.108	3.047	2.955	3.409	2.71	3.508	3.149	3.219
SHITOMASI BRISK	2.051	2.057	2.812	1.978	1.815	2.876	3.449	4.273	2.57	2.786	2.826	2.77	2.727	2.978	2.353	3.152	3.033	2.781
SHITOMASI FREAK	1.998	2.755	3.042	3.536	2.993	3.76	3.586	3.794	2.844	2.659	2.804	3.145	2.588	3.078	2.53	2.944	2.642	2.826
SHITOMASI ORB	2.5	3.351	3.148	3.233	3.094	3.883	2.803	4.2	2.996	2.948	2.816	2.937	3.002	3.369	2.593	3.366	3.202	3.623
SHITOMASI SIFT	2.5	3.103	3.053	2.408	2.877	4.143	2.803	4.551	3.34	2.981	2.795	2.949	2.997	3.295	2.895	2.806	3.072	3.419
SIFT BRIEF	4.283	3.822	3.914	5.204	5.059	3.615	4.022	3.914	2.658	2.287	3.293	2.673	2.399	2.527	2.458	2.036	1.343	1.696
SIFT BRISK	3.308	3.245	3.755	5.726	5.741	3.347	4.127	3.572	2.903	2.209	2.97	2.219	2.159	2.135	2.179	1.66	1.077	1.641
SIFT FREAK	4.504	3.042	2.863	4.918	5.469	3.591	3.854	4.268	2.527	1.894	2.91	2.209	2.065	2.268	1.731	1.21	0.9072	1.565
SIFT SIFT	3.534	3.785	2.838	5.142	3.903	2.853	3.659	4.129	3.122	2.493	2.753	2.593	2.088	2.388	2.179	1.833	1.017	1.187

frame number

Number of Inlier Matches Over Time

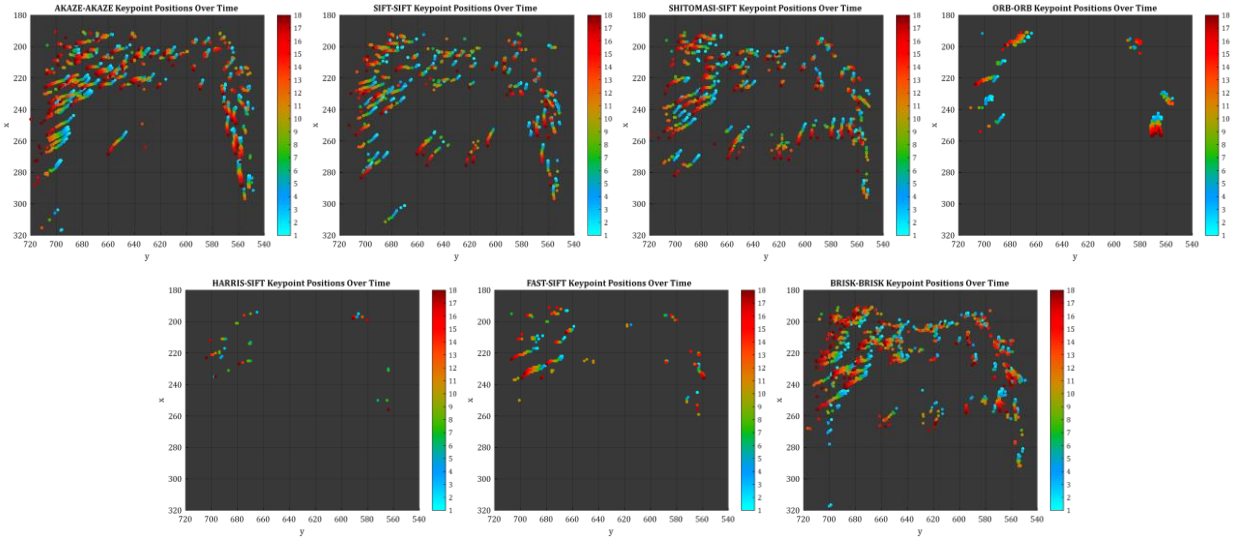
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
AKAZE AKAZE	68	77	74	68	71	73	83	86	86	96	89	90	80	66	61	49	52	71
AKAZE BRIEF	70	70	69	65	68	71	84	84	86	84	79	87	80	71	54	44	60	70
AKAZE BRISK	67	74	62	70	67	73	83	84	84	90	73	78	78	68	56	40	60	70
AKAZE FREAK	52	66	69	64	73	70	83	76	75	88	81	77	77	71	56	44	57	66
AKAZE ORB	52	70	67	59	71	63	86	82	81	88	81	78	75	64	55	33	57	63
AKAZE SIFT	73	77	81	69	73	74	87	88	86	97	89	92	85	82	64	50	64	65
BRISK BRIEF	62	60	73	51	58	46	72	59	82	82	80	76	76	56	64	33	59	47
BRISK BRISK	49	80	73	64	63	56	77	72	81	77	67	91	80	59	60	41	50	57
BRISK FREAK	50	59	38	51	49	41	51	62	76	68	64	72	56	48	64	40	44	51
BRISK ORB	65	75	73	50	59	51	74	68	77	84	82	50	71	57	62	33	49	61
BRISK SIFT	62	62	66	42	57	67	71	46	76	81	77	82	74	65	57	36	40	55
FAST BRIEF	8	9	4	7	9	8	13	6	16	18	17	14	12	11	15	11	10	0
FAST BRISK	7	6	4	7	7	6	9	12	14	17	11	13	9	11	12	8	8	6
FAST FREAK	4	5	4	7	7	6	7	7	14	11	12	9	8	10	12	10	8	6
FAST ORB	8	10	4	7	9	7	13	6	16	17	17	14	13	12	13	11	10	0
FAST SIFT	9	10	4	7	9	8	13	7	16	20	16	13	11	12	13	11	8	7
HARRIS BRIEF	1	1	0	2	7	2	7	5	0	1	2	2	4	0	2	6	0	0
HARRIS BRISK	1	0	0	2	5	0	5	5	0	0	2	2	4	0	2	5	0	0
HARRIS FREAK	1	0	0	2	6	2	6	3	0	1	2	2	2	0	1	4	0	0
HARRIS ORB	1	1	0	2	7	2	7	5	0	0	2	2	4	1	2	5	0	0
HARRIS SIFT	1	3	0	2	7	3	7	5	0	1	2	2	4	1	2	4	0	0
ORB BRIEF	8	15	14	9	10	12	14	10	14	12	9	10	9	10	0	8	10	2
ORB BRISK	16	24	21	16	25	23	23	15	21	20	17	19	22	17	17	14	13	9
ORB FREAK	8	15	9	9	11	12	7	0	8	8	5	6	9	7	0	0	0	0
ORB ORB	22	22	24	20	24	22	27	23	26	21	21	23	26	19	18	16	11	9
ORB SIFT	21	23	18	18	27	24	25	22	24	16	12	20	23	21	17	16	7	8
SHITOMASI BRIEF	48	52	38	48	39	47	61	62	57	61	53	64	47	36	46	34	24	47
SHITOMASI BRISK	46	32	37	38	42	46	48	50	49	52	37	57	52	38	35	23	32	31
SHITOMASI FREAK	44	40	34	41	50	48	50	49	44	47	43	50	45	39	34	27	28	31
SHITOMASI ORB	54	52	41	43	52	55	57	55	40	48	57	63	60	48	40	29	40	41
SHITOMASI SIFT	54	51	42	33	53	47	57	62	53	60	52	64	59	41	32	32	38	46
SIFT BRIEF	17	25	24	21	28	31	31	32	20	13	16	18	21	15	14	9	14	15
SIFT BRISK	21	20	24	13	14	31	28	31	18	14	18	16	21	15	15	11	9	14
SIFT FREAK	14	18	21	17	19	22	23	19	19	11	17	17	23	15	12	10	8	14
SIFT SIFT	27	31	27	29	33	36	35	40	47	19	20	21	27	20	17	14	12	20

frame number

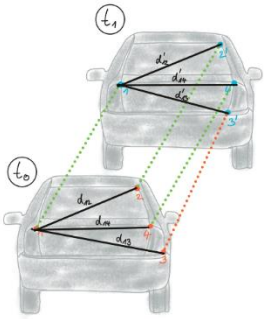
Based on the tables shown above, it appears that TTC estimation may be driven first by the detector type.

Figure 50: Keypoint Detector Types Evaluated in this Study

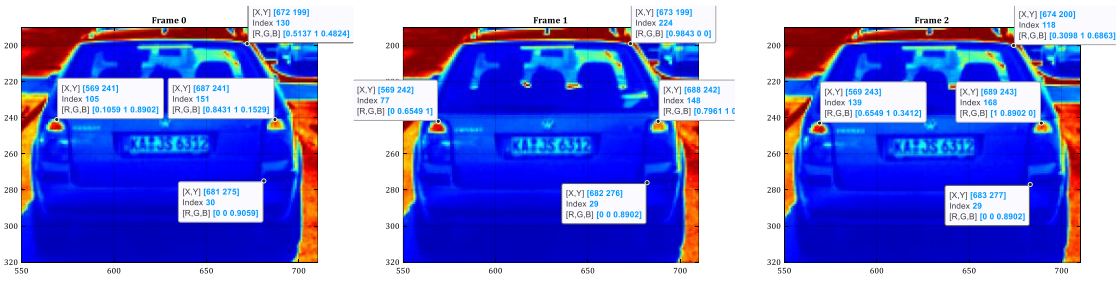
Seeking insights into what causes one detector method to perform better than others, I looked at the optical flow of a few combinations. In the following images, the keypoints in the bounding box are plotted over time, where the color gradient varies from Frame 0 (cyan) through Frame 18 (red):

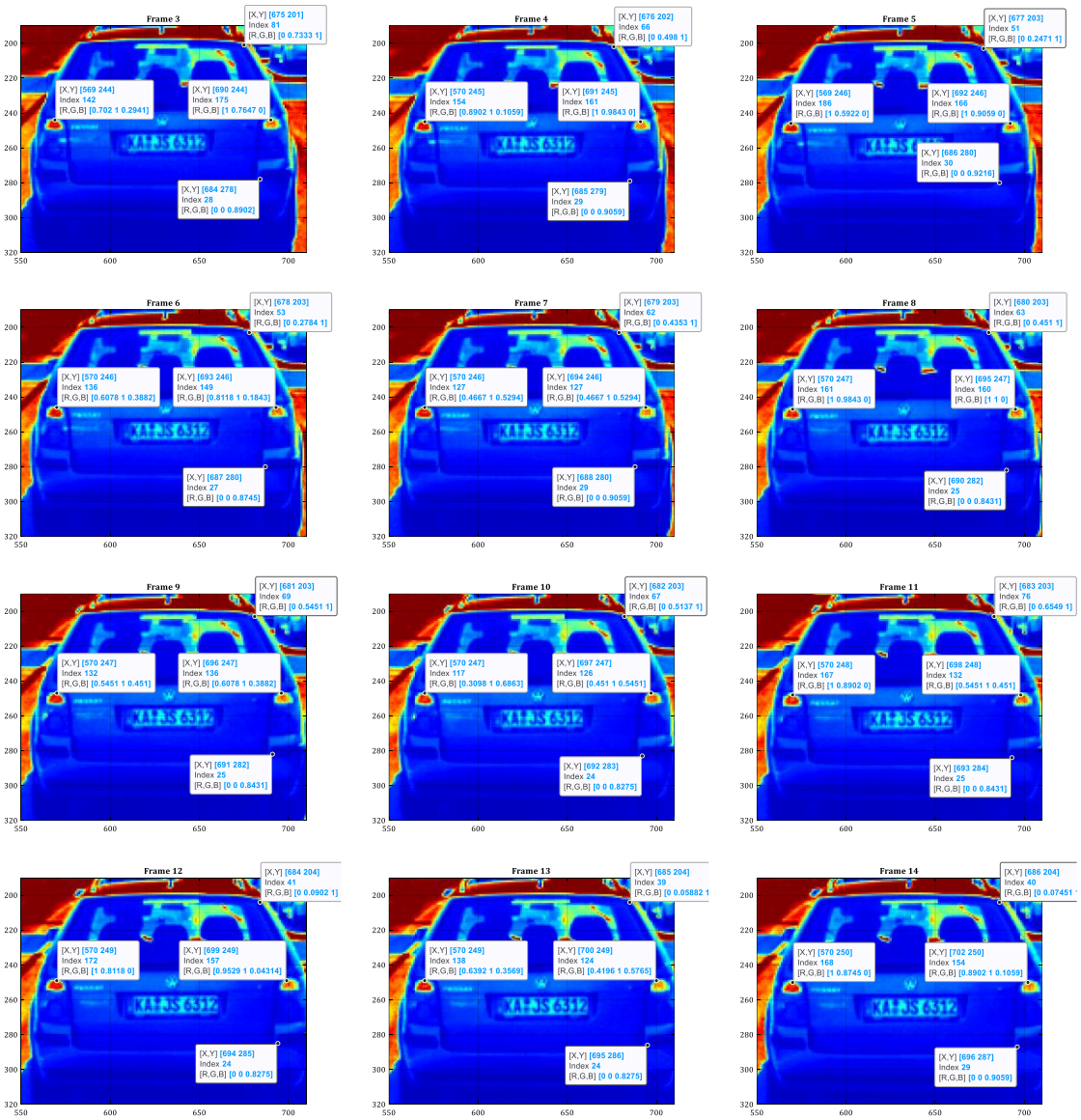


We lack a source of ground truth to evaluate performance, so I performed “eyeball” estimates of keypoints on all frames.



$$TTC = \frac{-\Delta t}{\left(1 - \frac{h_1}{h_0}\right)}$$




Frame 3
[X,Y] [675 201]
Index 81
[R,G,B] [0 0.7333 1]
[X,Y] [569 244]
Index 142
[R,G,B] [0.702 1 0.2941]
[X,Y] [690 244]
Index 175
[R,G,B] [1 0.7647 0]
[X,Y] [684 278]
Index 28
[R,G,B] [0 0 0.8902]
Frame 4
[X,Y] [676 202]
Index 66
[R,G,B] [0 0.498 1]
[X,Y] [570 245]
Index 154
[R,G,B] [0.8902 1 0.1059]
[X,Y] [691 245]
Index 161
[R,G,B] [1 0.9843 0]
[X,Y] [685 279]
Index 29
[R,G,B] [0 0 0.9059]
Frame 5
[X,Y] [677 203]
Index 51
[R,G,B] [0 0.2471 1]
[X,Y] [569 246]
Index 186
[R,G,B] [1 0.5922 0]
[X,Y] [692 246]
Index 166
[R,G,B] [1 0.9059 0]
[X,Y] [686 280]
Index 30
[R,G,B] [0 0 0.9216]
Frame 6
[X,Y] [678 203]
Index 53
[R,G,B] [0 0.2784 1]
[X,Y] [570 246]
Index 136
[R,G,B] [0.6078 1 0.3882]
[X,Y] [693 246]
Index 149
[R,G,B] [0.8118 1 0.1843]
[X,Y] [687 280]
Index 27
[R,G,B] [0 0 0.8745]
Frame 7
[X,Y] [679 203]
Index 62
[R,G,B] [0 0.4353 1]
[X,Y] [570 246]
Index 127
[R,G,B] [0.4667 1 0.5294]
[X,Y] [694 246]
Index 127
[R,G,B] [0.4667 1 0.5294]
[X,Y] [688 280]
Index 29
[R,G,B] [0 0 0.9059]
Frame 8
[X,Y] [680 203]
Index 63
[R,G,B] [0 0.451 1]
[X,Y] [570 247]
Index 161
[R,G,B] [1 0.9843 0]
[X,Y] [695 247]
Index 160
[R,G,B] [1 1 0]
[X,Y] [690 282]
Index 25
[R,G,B] [0 0 0.8431]
Frame 9
[X,Y] [681 203]
Index 69
[R,G,B] [0 0.5451 1]
[X,Y] [570 247]
Index 132
[R,G,B] [0.5451 1 0.451]
[X,Y] [696 247]
Index 136
[R,G,B] [0.6078 1 0.3882]
[X,Y] [691 282]
Index 25
[R,G,B] [0 0 0.8431]
Frame 10
[X,Y] [682 203]
Index 67
[R,G,B] [0 0.5137 1]
[X,Y] [570 247]
Index 117
[R,G,B] [0.3098 1 0.6863]
[X,Y] [697 247]
Index 126
[R,G,B] [0.451 1 0.5451]
[X,Y] [692 283]
Index 24
[R,G,B] [0 0 0.8275]
Frame 11
[X,Y] [683 203]
Index 76
[R,G,B] [0 0.6549 1]
[X,Y] [570 248]
Index 167
[R,G,B] [1 0.8902 0]
[X,Y] [698 248]
Index 132
[R,G,B] [0.5451 1 0.451]
[X,Y] [693 284]
Index 25
[R,G,B] [0 0 0.8431]
Frame 12
[X,Y] [684 204]
Index 41
[R,G,B] [0 0.0902 1]
[X,Y] [570 249]
Index 172
[R,G,B] [1 0.8118 0]
[X,Y] [699 249]
Index 157
[R,G,B] [0.9529 1 0.04314]
[X,Y] [694 285]
Index 24
[R,G,B] [0 0 0.8275]
Frame 13
[X,Y] [685 204]
Index 39
[R,G,B] [0 0.05882 1]
[X,Y] [570 249]
Index 138
[R,G,B] [0.6392 1 0.3569]
[X,Y] [700 249]
Index 124
[R,G,B] [0.4196 1 0.5765]
[X,Y] [695 286]
Index 24
[R,G,B] [0 0 0.8275]
Frame 14
[X,Y] [686 204]
Index 40
[X,Y] [570 250]
Index 168
[R,G,B] [1 0.8745 0]
[X,Y] [702 250]
Index 154
[R,G,B] [0.8902 1 0.1059]
[X,Y] [696 287]
Index 29
[R,G,B] [0 0 0.9059]

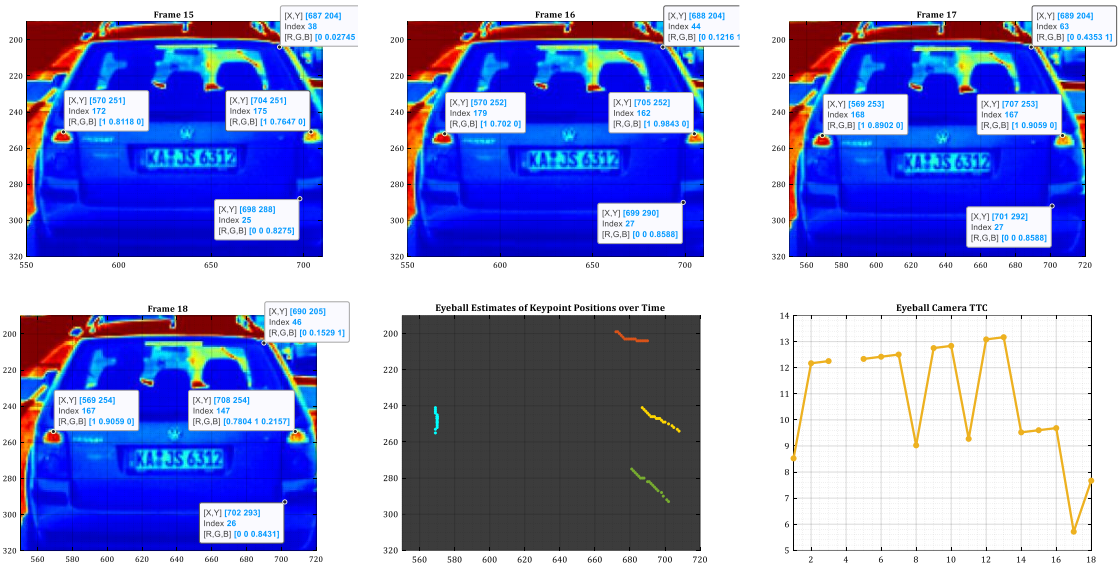


Figure 51: Manual “Eyeball” Estimate of the Camera Keypoint-Based TTC

I also looked for multi-sensor (camera + lidar) consensus / similarities as a surrogate for truth. Given these results, I determined that the group of AKAZE detectors produced temporal TTC trends most similar to the lidar-based TTC estimates.

Figure 52: Example of AKAZE Keypoints

In the graph below, I grouped the AKAZE keypoint results by taking the median TTC across all corresponding descriptor methods, at every frame in time.

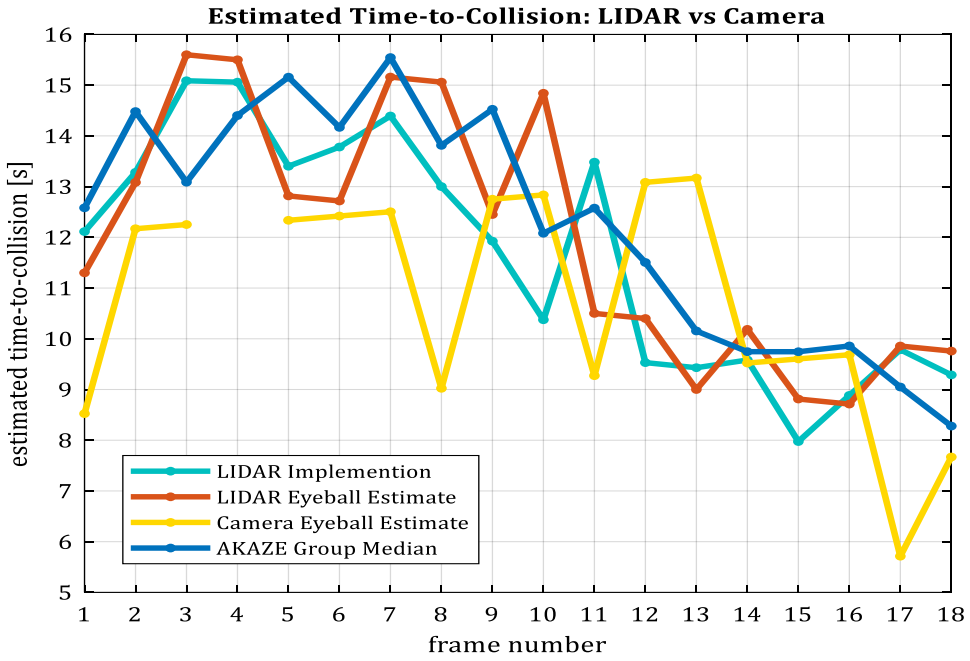
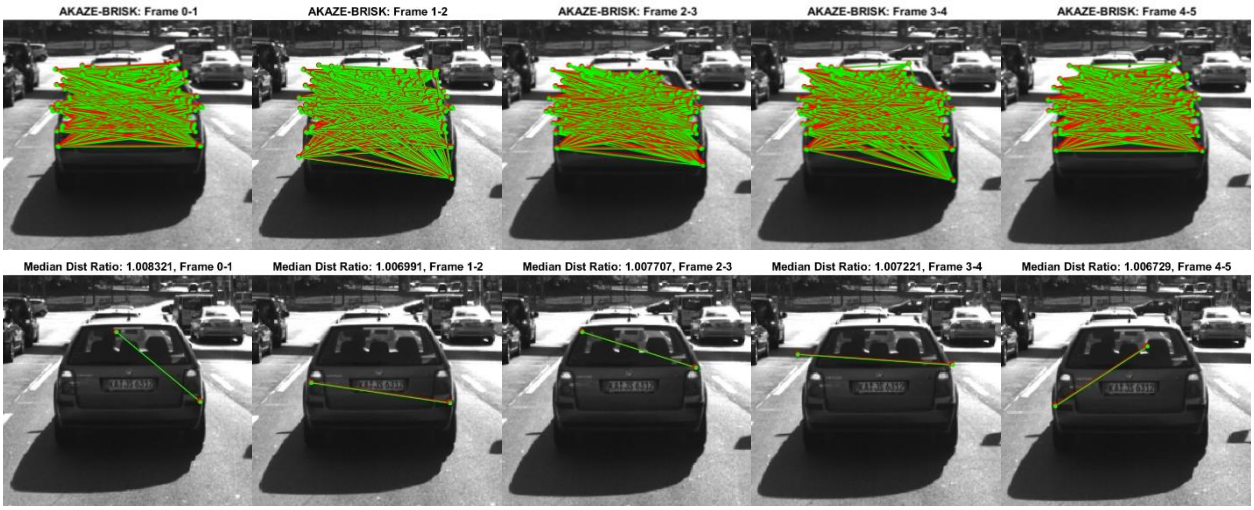


Figure 53: Estimated Time-to-Collision: LIDAR vs Camera

AKAZE-BRISK

I chose AKAZE detectors with BRISK descriptors as a configuration to look at because it produced trends with the same order of magnitude as the LIDAR TTC results. The following images show the keypoints used to estimate TTC on each frame (row 1), followed by the keypoint pairs corresponding to the median distance ratio (row 2). Points and lines in red correspond to the previous frame, while points and lines in green correspond to the current frame:



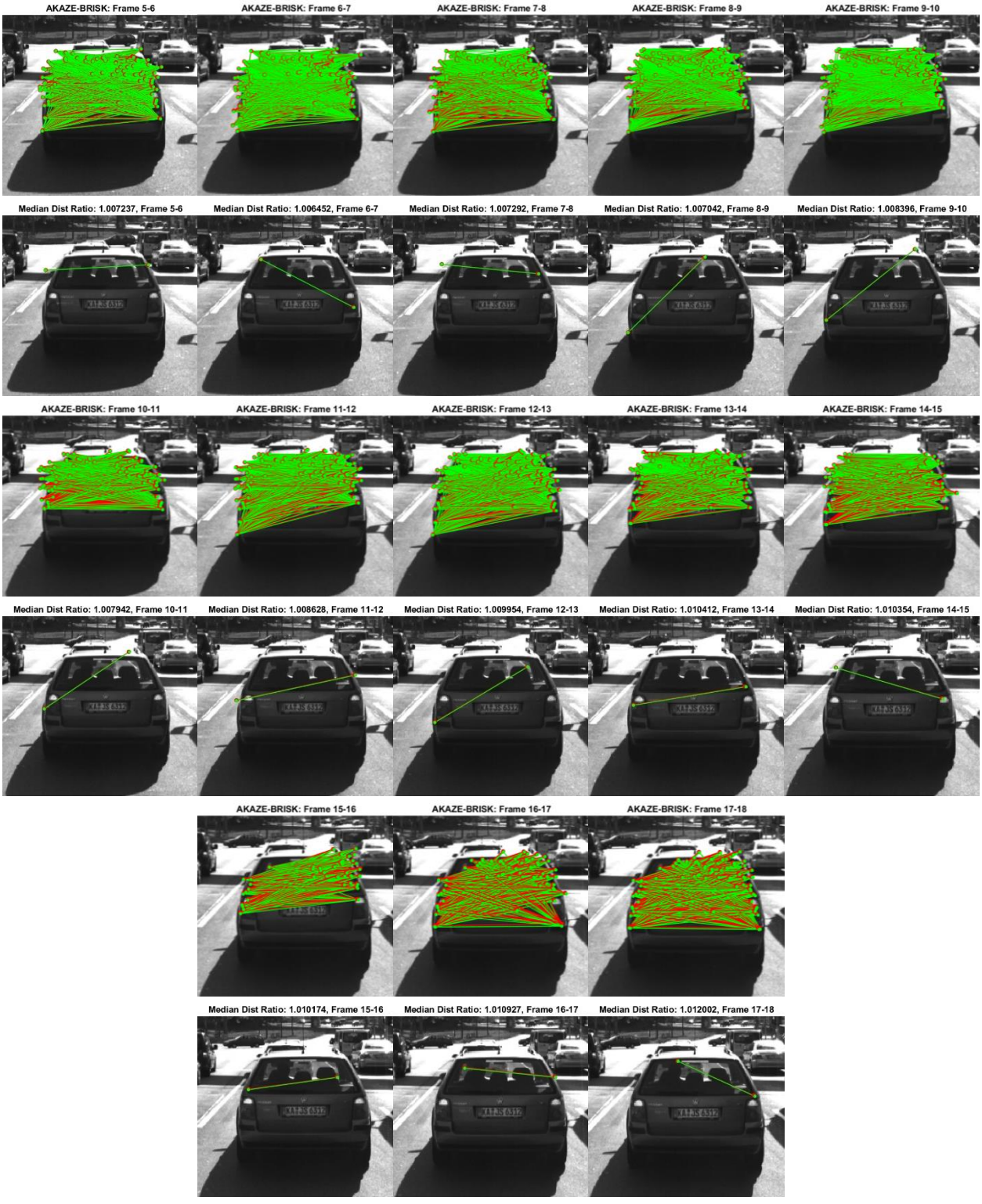


Figure 54: Keypoints Corresponding to the Median Distance Ratio for TTC Estimation

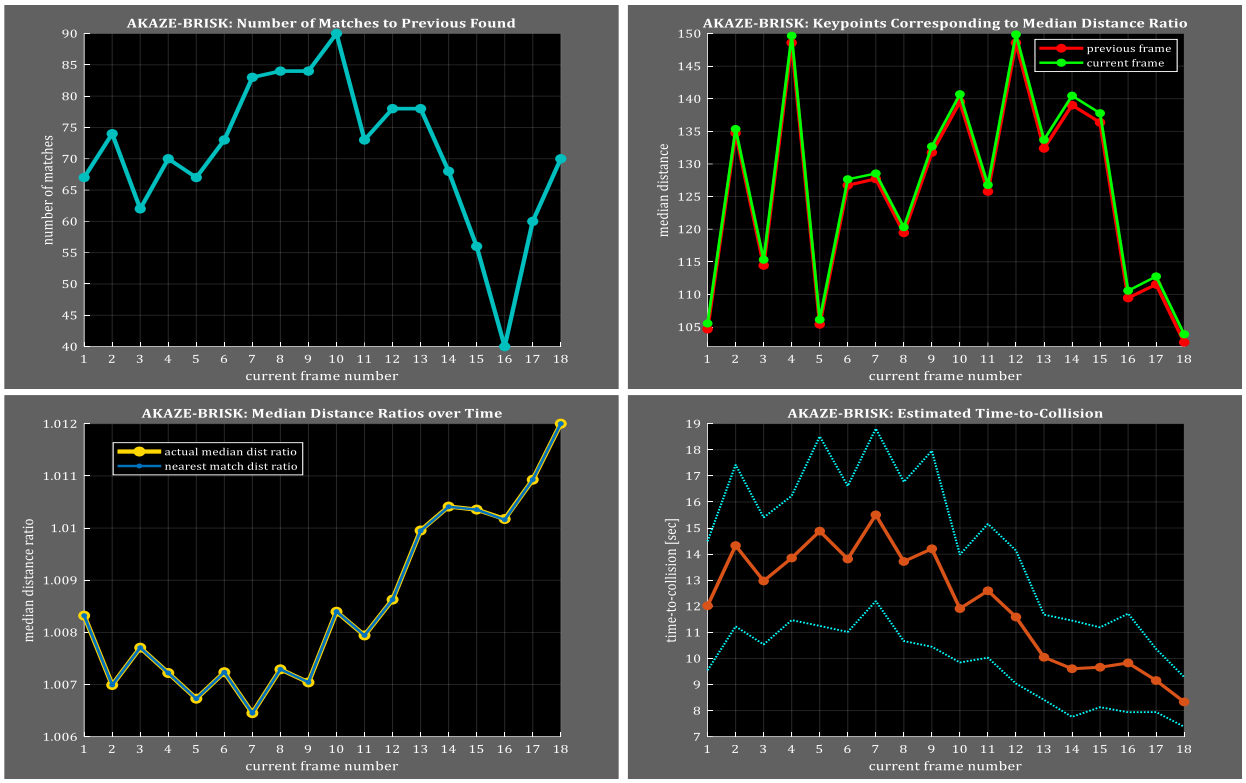


Figure 55: Median Distance Ratio and Estimated TTC based on AKAZE Detector & BRISK Descriptor

On the other end of the spectrum, we looked into why some approaches produced TTC estimates that were way off.

HARRIS-SIFT

The HARRIS detection method produced a very sparse set samples from which we could compute the TTC. Recall the multiple entry criteria that were required for samples to qualify for TTC estimation:

First, Harris Corners were detected in the monochrome image:

Figure 56: Harris Corner Keypoints Detected on Frame 7

Second, when keypoints were found, they had to be matched with keypoints in adjacent frames:

Figure 57: Keypoint Matches Between Frames 6 and 7

Third, keypoints not associated with a YOLO bounding box were excluded. Also, keypoints that fell within more than one bounding box were removed:

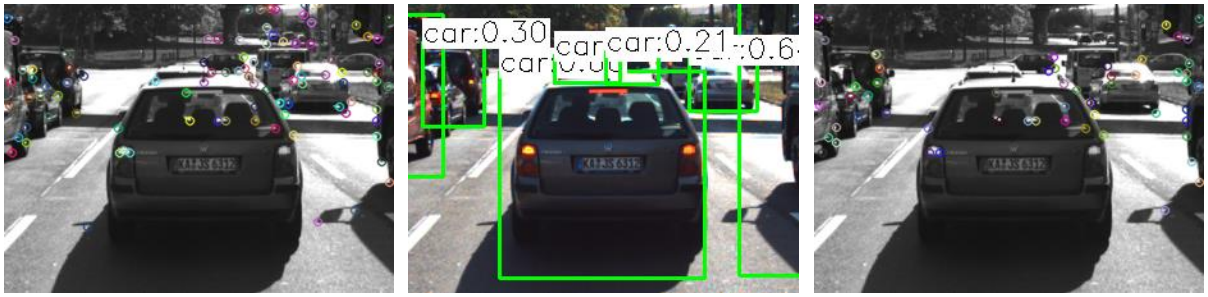
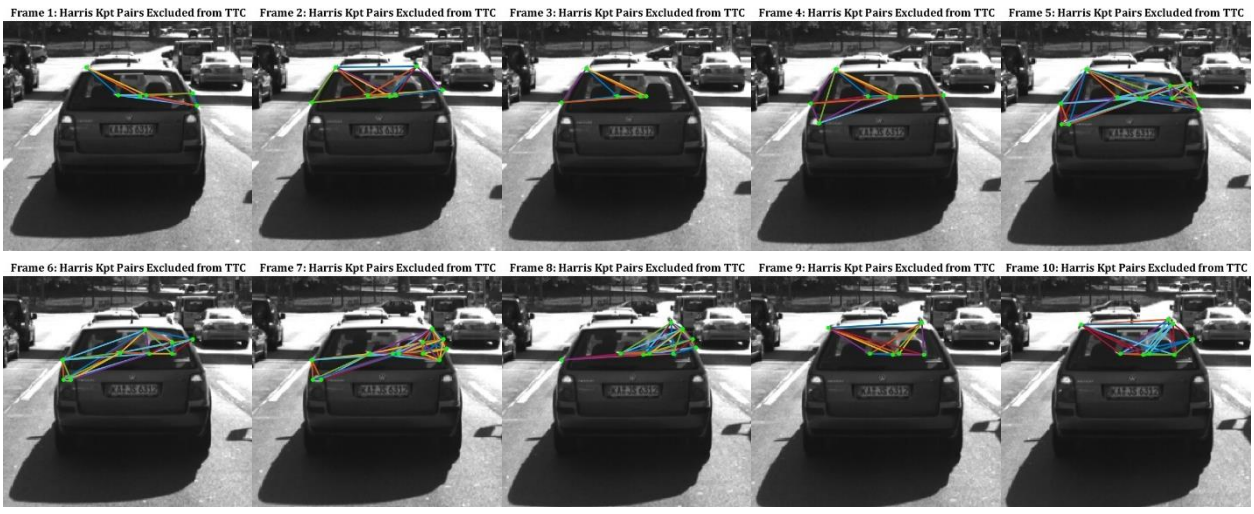


Figure 58: Before (far left) and after (far right) removing keypoints associated with multiple bounding boxes

Fourth, only keypoints within the bounding box of interest* were looked at (*lidar constraints applied earlier usually narrowed it down to one box).

Fifth, intraframe keypoint pairings were required to have a distance greater than or equal to 100cm. The intent was to capture the more distal keypoint pairs stretching across the extent of the vehicle, which would provide better measurements for detecting a change in the projected area than keypoint pairs that were closer together.

The following figures show the HARRIS detection pairs that were excluded from the TTC estimation for having relative distances below 100cm. Note that the filtering criteria has the effect of removing inner keypoints from consideration:



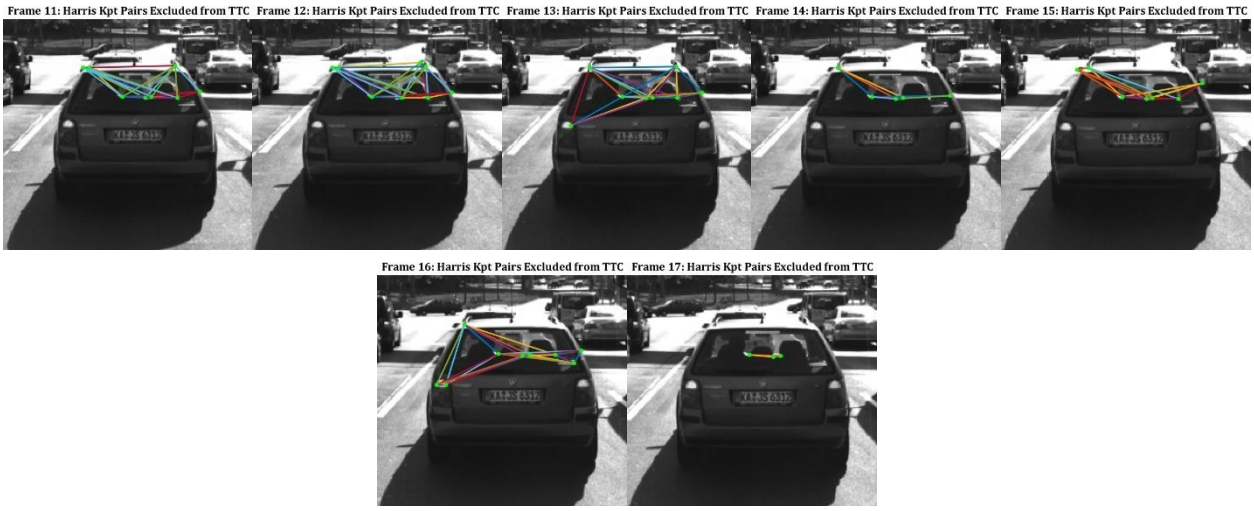


Figure 59: Harris Keypoint Pairs with Distance less than 1m (and therefore excluded from TTC estimation)

Ideally, we would emerge from these 5 stringent entry criteria with a statistically significant number of samples to produce a median distance ratio that captures the frame-to-frame motion.

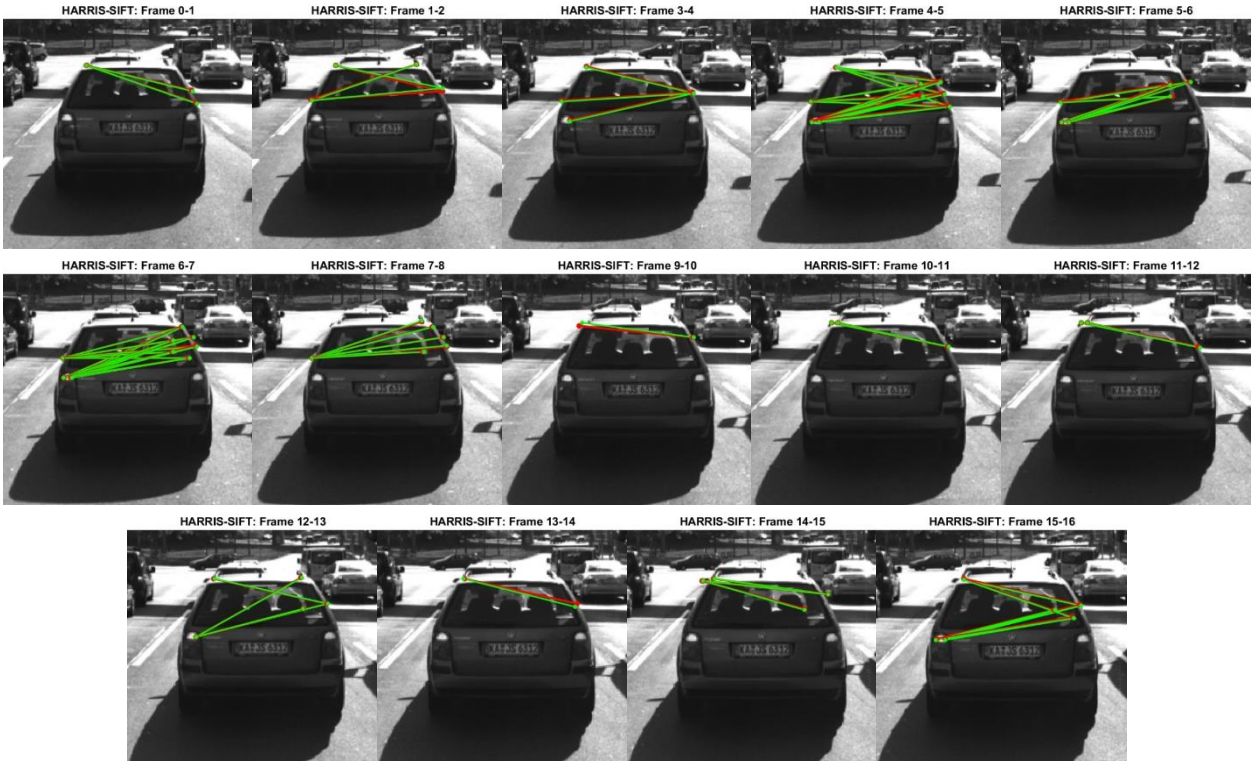


Figure 60: Keypoint pairs with distance greater than 1m: HARRIS Detector - SIFT Descriptor

For all keypoint pairings that manage to pass the stringent list of entry criteria, only the median distance ratio is used to compute the TTC. The more uniform this filtered distribution is, the better the chances of

pulling a good representation. In the case of HARRIS detectors, the small number of surviving samples leaves little room for error in keypoint position accuracy:

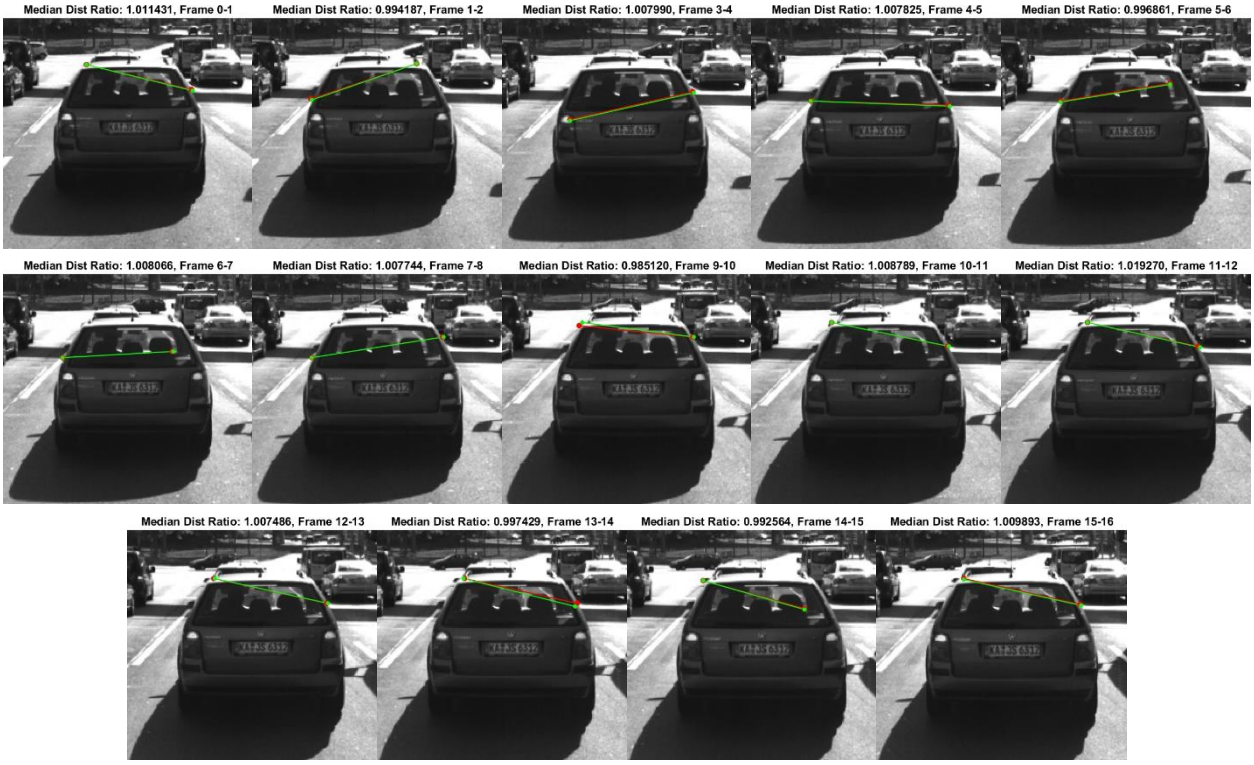


Figure 61: Keypoint Pairs Corresponding to the Median Distance Ratio

If no matching keypoints between a current and previous frame were found within the bounding box, then a TTC could not be computed on the current frame. The figure below shows that we had single-frame TTC dropouts on frames 3 and 9, indicating that no matching keypoints were found in the bounding box between frames 2-3 and 8-9. In addition, both frames 17 and 18 did not produce TTC estimates, which shows that one or both may not have had any keypoints at all.

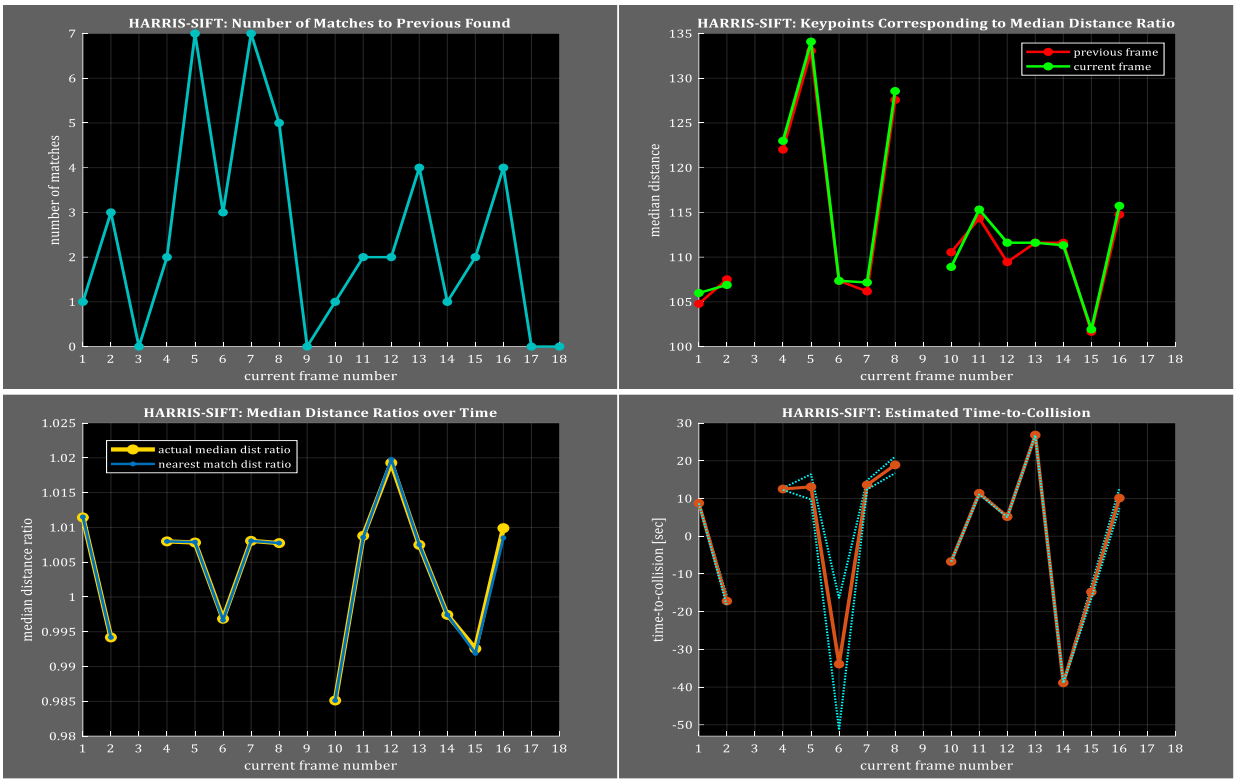


Figure 62: Median Distance Ratio and Estimated TTC based on HARRIS Detector & SIFT Descriptor

Frames 2, 6, 10, 14 and 15 all produced negative TTC estimates, suggesting that the distance between the ego car and preceding vehicle have increased between frames. While that may be the case, these estimates were based on a very small number of sample matches – or 3, 3, 1, 1 and 2 matches, respectively. The keypoints corresponding to the median distance ratio would need to have a very high level of accuracy and consistency in identifying the same point on *both* of the frames being compared. The images below show how small that margin of error is:

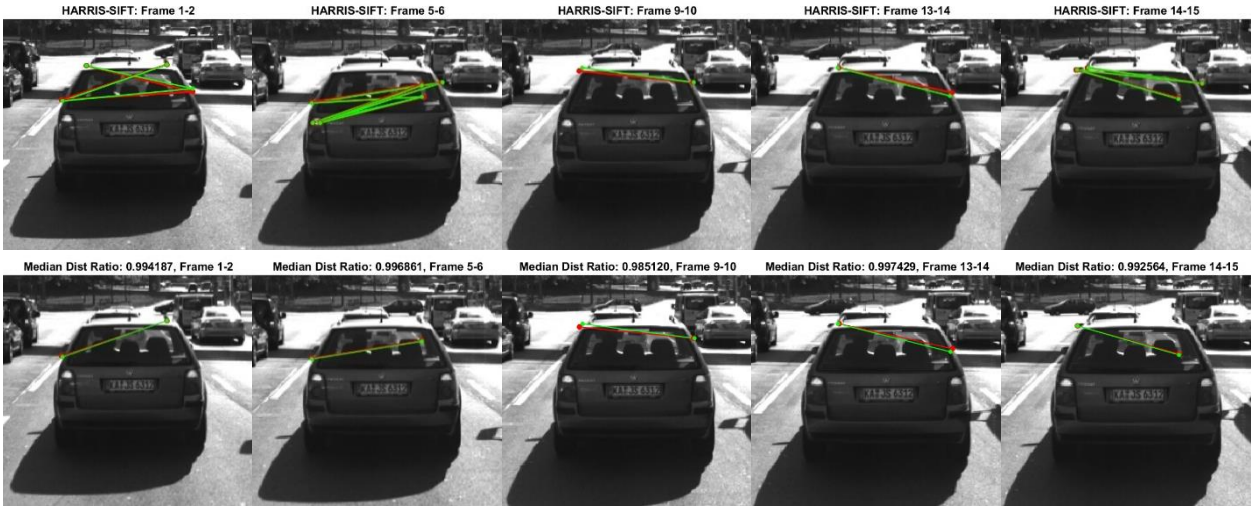


Figure 63: Frames with negative TTC: top row shows all keypoint pairs, bottom row shows median dist ratio pair

ORB-SIFT

ORB detectors produce a large number of keypoints over a relatively small number of distinct locations. Each location consists of a group of concentric circles that represent collection regions of varying size, scale and orientation. These features may prove helpful in keypoint matching, but we still end up with redundant matches within groups.

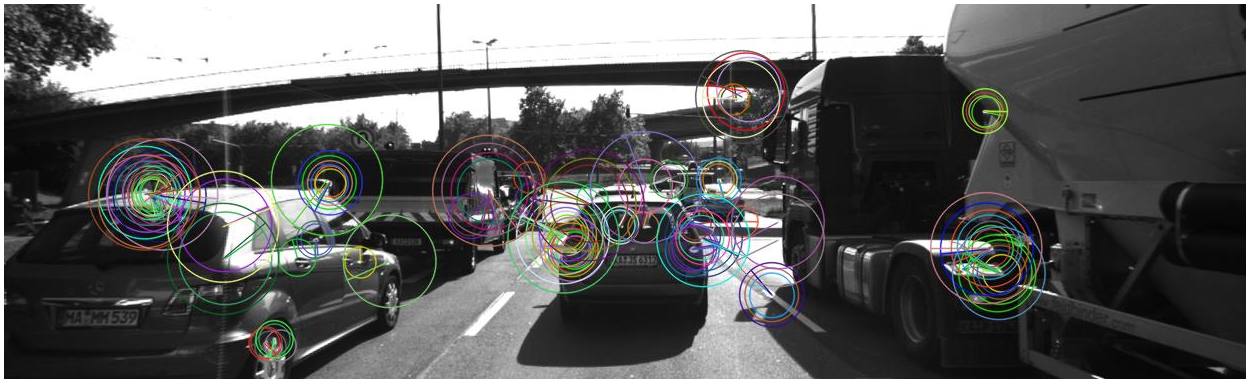


Figure 64: ORB keypoints that were associated with exactly one bounding box on Frame 3 (overlaps removed)

Our camera-based TTC algorithm currently only uses the centroid information from keypoints, effectively collapsing the multi-scale regions into groups of matches with small variations in keypoint positions. This could present a risk factor for our estimation approach, where the smallest errors in keypoint position can be amplified at the TTC estimation level.

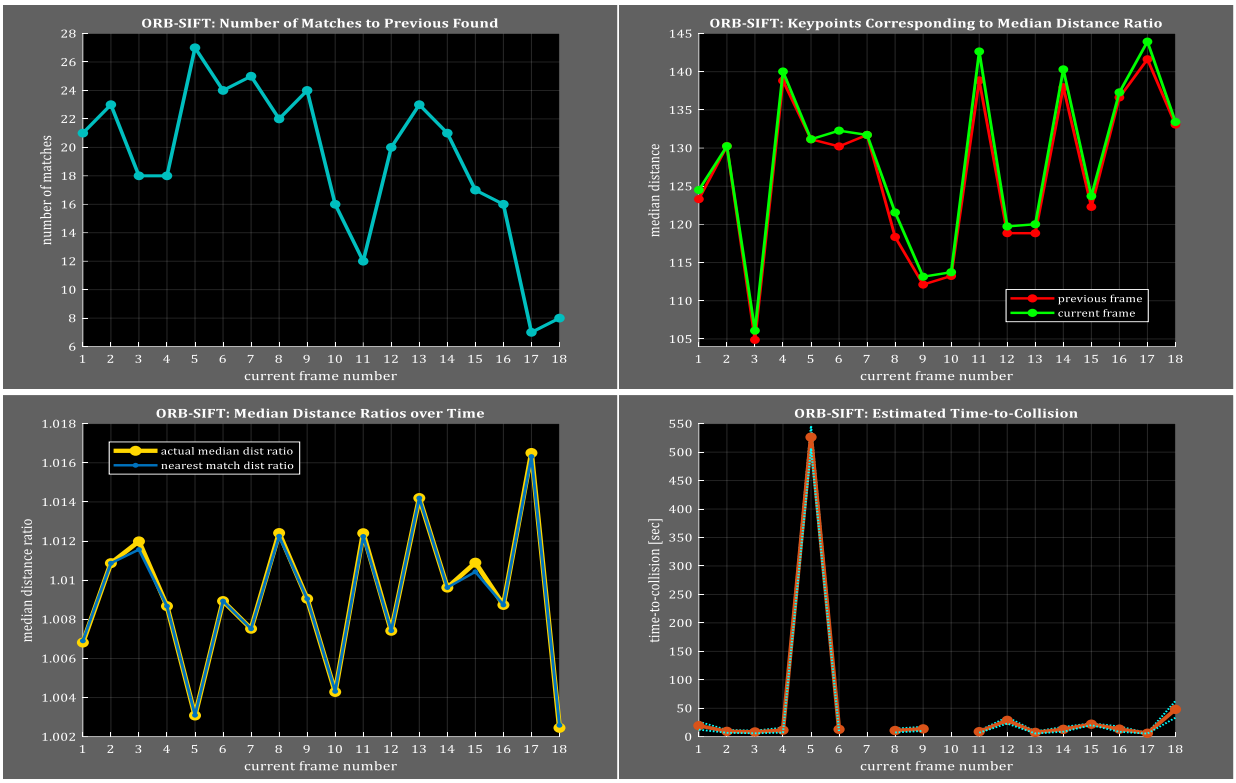
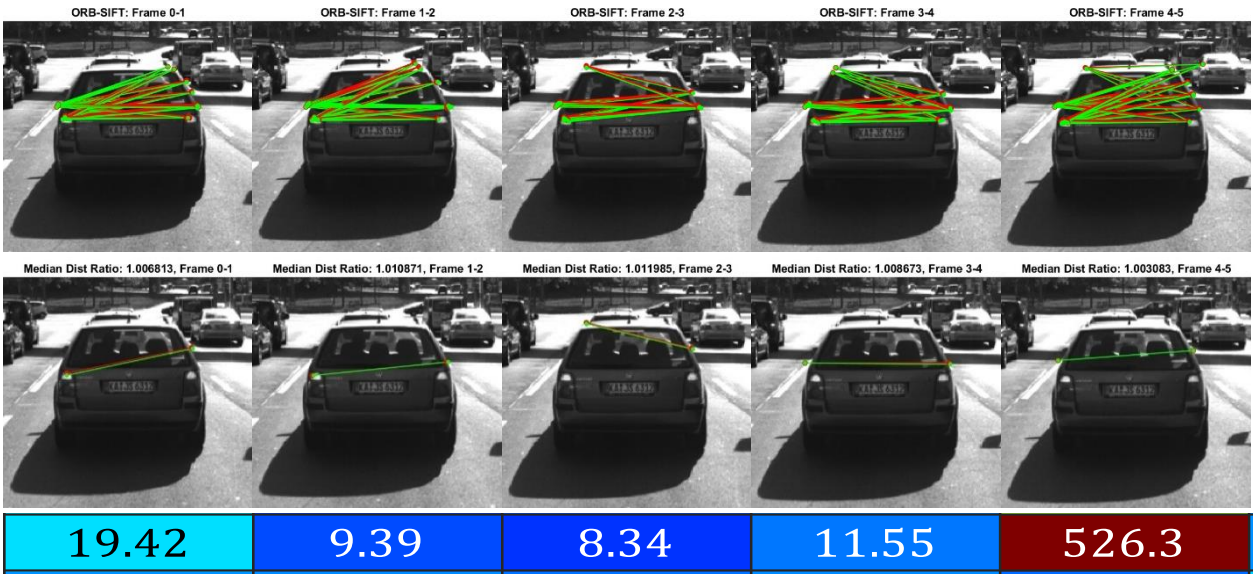
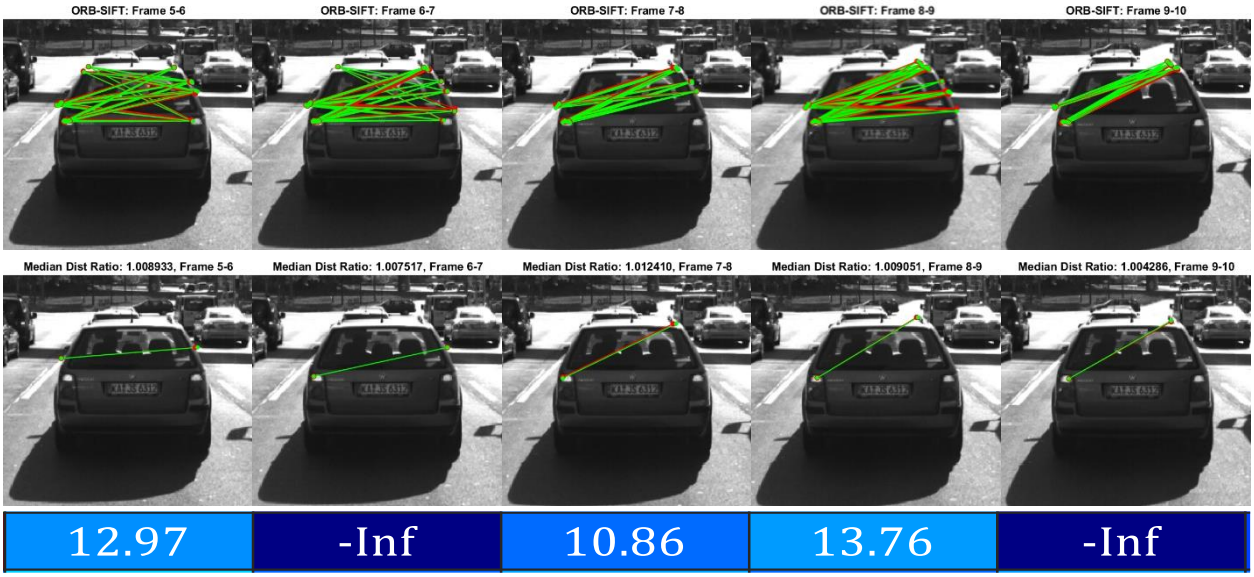


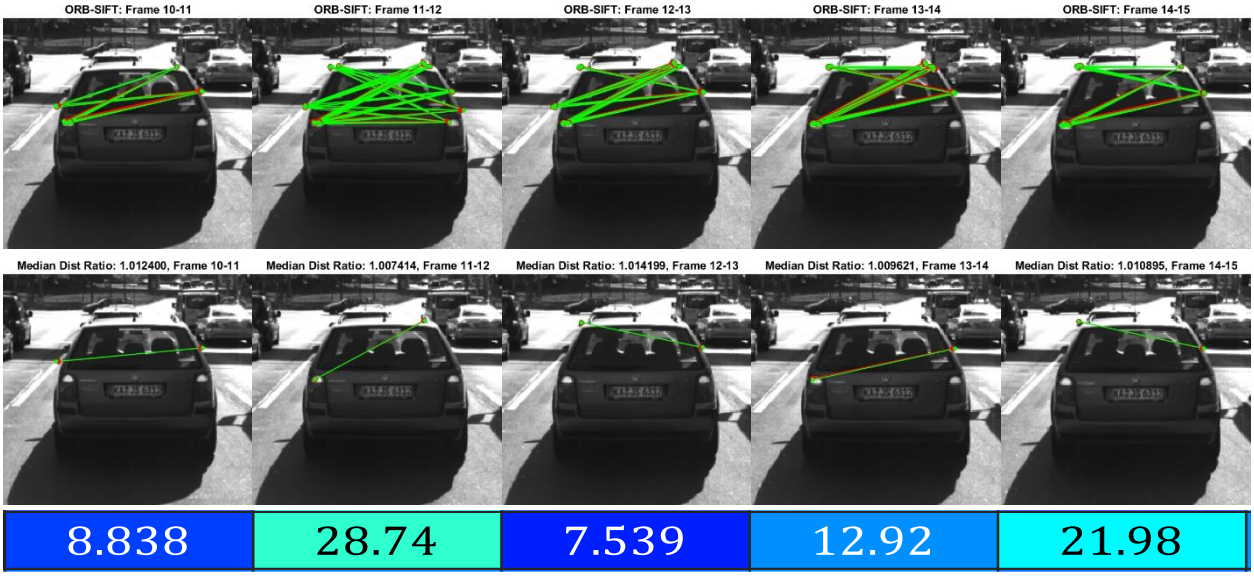
Figure 65: Median Distance Ratio and Estimated TTC based on ORB Detector & SIFT Descriptor

ORB SIFT	19.42	9.39	8.34	11.55	526.3	12.97	-Inf	10.86	13.76	-Inf	8.838	28.74	7.539	12.92	21.98	13.28	6.059	48.08



19.42	9.39	8.34	11.55	526.3


ORB-SIFT: Frame 5-6
ORB-SIFT: Frame 6-7
ORB-SIFT: Frame 7-8
ORB-SIFT: Frame 8-9
ORB-SIFT: Frame 9-10
Median Dist Ratio: 1.008933, Frame 5-6
Median Dist Ratio: 1.007517, Frame 6-7
Median Dist Ratio: 1.012410, Frame 7-8
Median Dist Ratio: 1.009051, Frame 8-9
Median Dist Ratio: 1.004286, Frame 9-10
12.97
-Inf
10.86
13.76
-Inf


ORB-SIFT: Frame 10-11
ORB-SIFT: Frame 11-12
ORB-SIFT: Frame 12-13
ORB-SIFT: Frame 13-14
ORB-SIFT: Frame 14-15
Median Dist Ratio: 1.012400, Frame 10-11
Median Dist Ratio: 1.007414, Frame 11-12
Median Dist Ratio: 1.014199, Frame 12-13
Median Dist Ratio: 1.009621, Frame 13-14
Median Dist Ratio: 1.010895, Frame 14-15
8.838
28.74
7.539
12.92
21.98

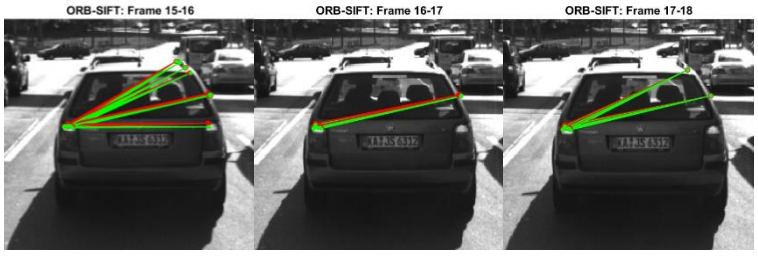

ORB-SIFT: Frame 15-16
ORB-SIFT: Frame 16-17
ORB-SIFT: Frame 17-18

Figure 66: Keypoint Pairs Corresponding to the Median Distance Ratio & TTC

I selected frames 5, 7, 10 and 18 below as examples of cases for which the TTC estimates seemed way off. We observe how ORB detectors amass a large number of keypoints, but those keypoints are clustered around only ~3-8 distinct feature points on the vehicle (and sometimes erroneous points outside of the vehicle). Nonetheless, there are enough observable offsets in the groups of red (previous frame) vs green (current frame) distance pairs in the first row of images below that we can actually visualize the motion from frame-to-frame. The problem then arises when the single median distance ratio selected from these distributions (second row) turns out to be a poor representation of the motion observed in the aggregate sample distribution (first row):

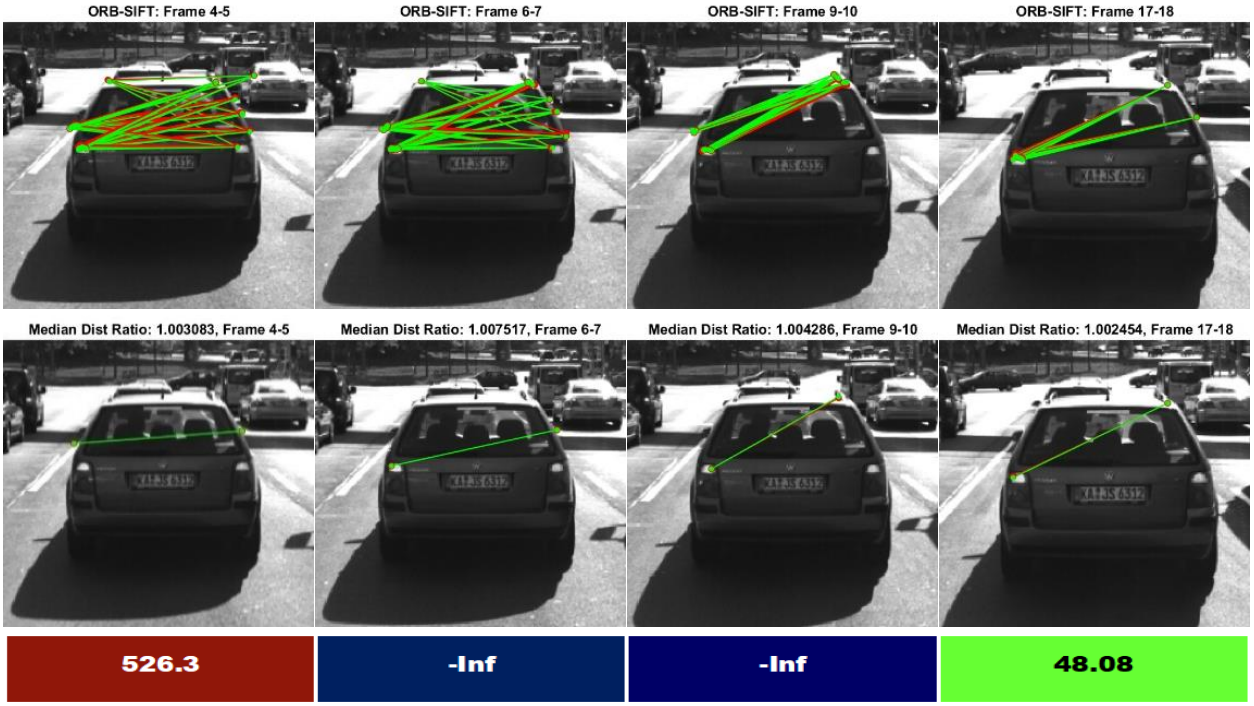


Figure 67: Top Row: Distal Keypoint Pairs, Middle Row: Median Distance Pair, Bottom: Estimated Time-to-Collision

Frames 4-5, 6-7 and 9-10 above seemed to “fail” because almost no motion was detected based on the median distance ratio and corresponding previous and current keypoint positions – and our TTC equation assumes constant velocity during these time intervals. These may have been unlucky draws, because the

offsets can be observed when looking at the full sets of distance ratios. To mitigate these cases, we may want to look at different approaches to estimating/describing the distance ratio, or possibly increasing the temporal gap between successive frames to increase the likelihood of selecting keypoints with observable changes in distance ratio.

Frames 4 & 5 also appear to have selected a keypoint that belongs to a while lane marking on the left-hand side, which would also corrupt our TTC estimate on the preceding vehicle. This issue of keypoints picking up features outside of the vehicle may be happening in other cases as well.

Performance is also dependent upon the detector-descriptor approach used, since inaccuracies at the finer granularities are amplified by the constant-velocity TTC equation. In the case of ORB, many keypoint candidates are presented in the vicinity of salient features, but errors are introduced when we try to collapse the larger collection regions into a single point.