Project Specification

Track an Object in 3D Space

FP.0 Final Report
Provide a Write up / README that includes all the rubric points and how you addressed each one. You can submit your write up as markdown or PDF.

→ This Document

FP.1 Match 3D Objects
Implement the method "matchBoundingBoxes", which takes as input both the previous and the current data frames and provides as output the ids of the matched regions of interest (i.e. the boxID property). Matches must be the ones with the highest number of keypoint correspondences.

This rubric point is addressed in the following function from camFusion_Student.cpp
Line No 203 int **matchBoundingBoxes**(boost::circular_buffer<DataFrame> *dataBuffer)
This function takes in the pointer to the dataBuffer containing prev and current dataFrame. Will iterate through the keypoint matches from current frame , extract the keypoints from the match find which boundingbox the keypoint belongs and put corresponding match in the std::vector<CV::DMatch> kptMatches for that bounding box.
Simultaneously it keeps a track of corresponding match counts in a 2D array for each bounding box from previous frame to all bounding boxes in current frame. Later each bounding box from prev frame is associated with a box from current frame where the keypoint match counts are maximum.

```
 int prev_to_curr[(dataBuffer->end()-2)->boundingBoxes.size()][(dataBuffer->end()-
1)->boundingBoxes.size()];
```

FP.2 Compute Lidar-based TTC
Compute the time-to-collision in second for all matched 3D objects using only Lidar measurements from the matched bounding boxes between current and previous frame.

This rubric points are addressed in the following two function from camFusion_Student.cpp

```
Line No 194 float computeTTCLidar(double sensorFrameRate, BoundingBox
*prevBB,BoundingBox *currBB)

Line No 171 float IQR_median(std::vector<LidarPoint> &lidarPoints)
```

→ Write Up

The IQR_median function uses interquartile range (IQR) method to remove some of the the outliers in the lidarPoints data set. The median of the resulting data set is used to find the distance to the preceding vehicle.
Below motion model Figure 1 and Figure 2 is used to calculate the lidar based ttc. This approach gives sufficiently reliable results as discussed in the performance section below.

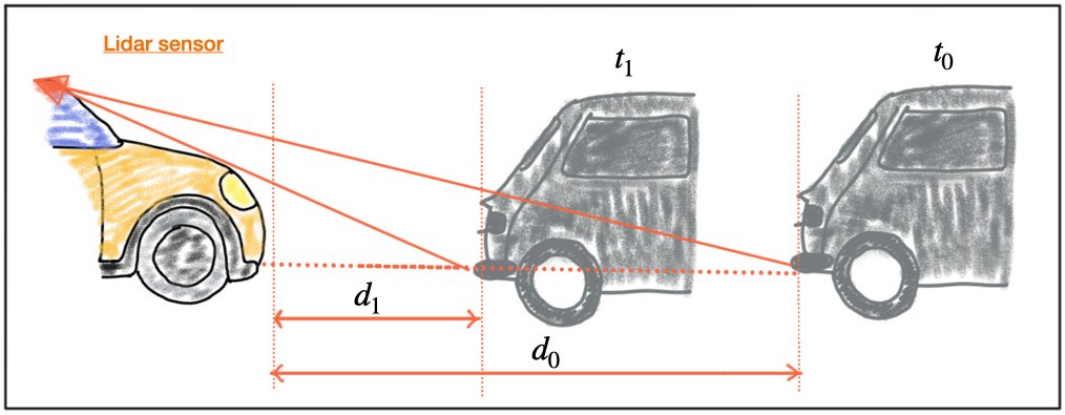

Figure 1

$$(1) \quad d(t+\Delta t) = d(t) - v_0 \cdot \Delta t$$

$$(2) \quad v_0 = \frac{d(t) - d(t+\Delta t)}{\Delta t} = \frac{d_0 - d_1}{\Delta t}$$

$$(3) \quad TTC = \frac{d_1}{v_0} = \frac{d_1 \cdot \Delta t}{d_0 - d_1}$$

Figure 2

FP.3 Associate Keypoint Correspondences with Bounding Boxes

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

→ Write up

camFusion_Student.cpp Line No 142

```
void clusterKptMatchesWithROI(BoundingBox &boundingBox)
```

Takes in the previously clustered kptmatches for the given bounding box and implements filtering for the Dmatch.distance. All the points where the distance higher than the 0.7*median distance are eliminated as outliers.

```
DMatch.distance - Distance between descriptors. Lower distances mean more accurate the match.
```

FP.4 Compute Camera-based TTC

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

→ Write Up

This rubric point is addressed in the following function from camFusion_Student.cpp

```
Line No 146 float computeTTCCamera(double
sensorFrameRate,boost::circular_buffer<DataFrame> *dataBuffer,BoundingBox
*prevBB,BoundingBox *currBB)

Line No 151 float IQR_median(vector<double> &distRatios)
```

Camera-based ttc algorithm used here is from the Udacity Lesson 3 “Estimating TTC with Camera”. The bounding box matches where we have lidar points are selected for ttc estimate which happens to be our ego lane.

The ratios of all the relative distances between keypoints from prev frame to current frame are calculated. The resulting dataset is passed to the IQR_median function which uses interquartile range (IQR) method to remove some of the the outliers in the distances data set. The median of the resulting data set is returned back to **computeTTCCamera to** find the distance to the preceding vehicle.

Following Figure 3 and Figure 4 camera based motion model is used to calculate the TTC.

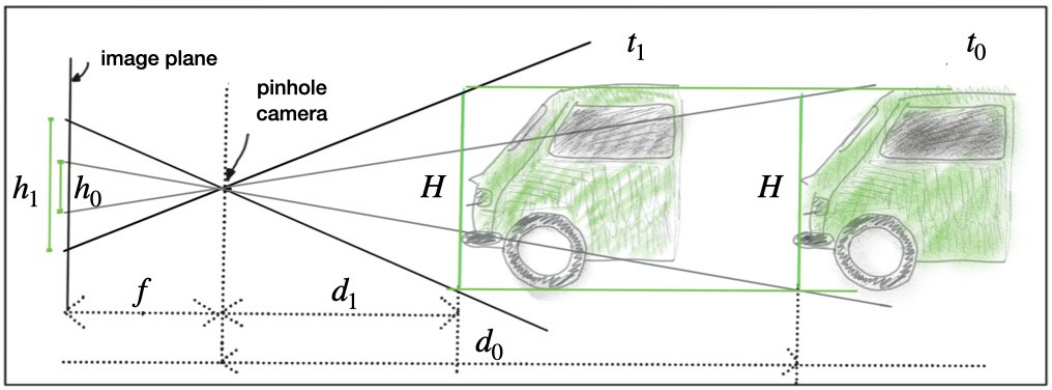

Figure 3

project object into camera

$$(1)\quad h_0 = \frac{f \cdot H}{d_0};\quad h_1 = \frac{f \cdot H}{d_1}$$

relate projection and distance

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

substitute in constant-velocity model

$$(3)\quad d_1 = d_0 - v_0 \cdot \Delta t = d_1 \cdot \frac{h_1}{h_0} - v_0 \cdot \Delta t$$

$$\rightarrow\quad d_1 = \frac{-v_0 \cdot \Delta t}{\left(1 - \frac{h_1}{h_0}\right)}$$

compute time to contact / collision

$$(4)\quad TTC = \frac{d_1}{v_0} = \frac{-\Delta t}{\left(1 - \frac{h_1}{h_0}\right)}$$

Figure 4

FP.5 Performance Evaluation 1

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

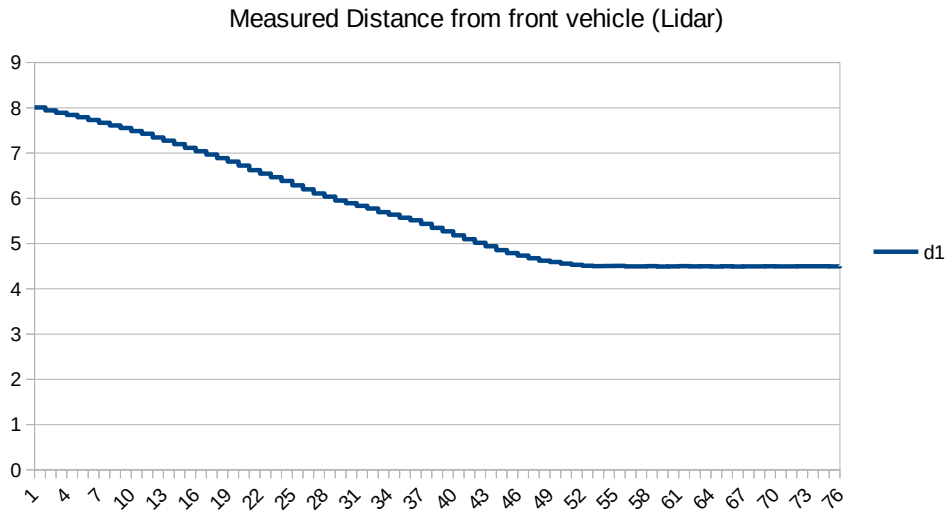

Figure 5

Figure 5 shows the measured Lidar distance d1 from the front vehicle. Distance measurement seems reliable enough. Looks like the vehicles are approaching a traffic light and stopping after frame no 50. The supporting data is available in Data/LidarTTC.csv file.

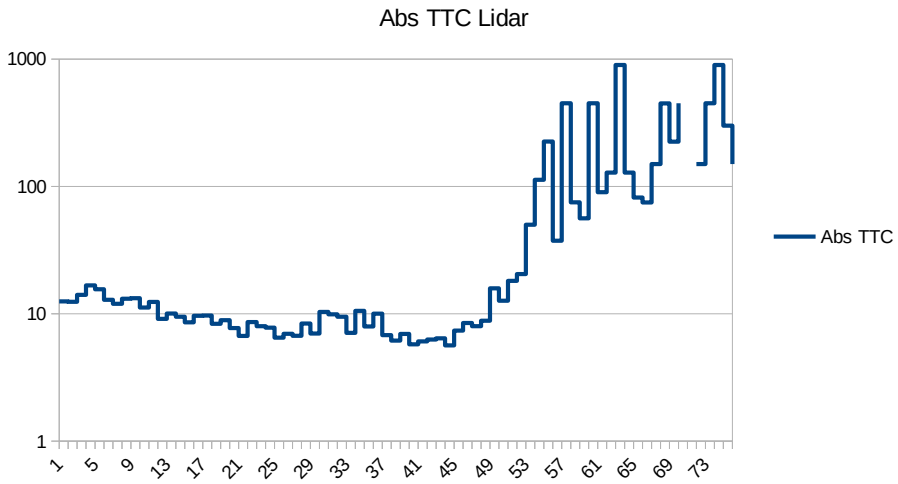

Figure 6

Figure 6 shows the lidar abs TTC estimates over all 77 frames. The ttc estimates seems reasonable till frame no 50, above which the numbers get very large and negative as well. Although this may seem inaccurate , this is expected as the vehicles are stopped and the TTC estimate will be a large number. The (d0-d1) value becomes very very small. The negative numbers are due to measured d1 being larger then d0 although by very small value (within tolerance of lidar measurements).

Inaccuracy in Lidar TTC -
The Lidar Motion model used here d1 * (1.0 / sensorFrameRate) / (d0 – d1) will give negative results if the measured d1 > d0. These results are inaccurate and the motion model needs to be refined to consider this situation.
Some of the spill over of lidar points to wrong bounding boxes eg truck on right side which can lead to wrong Lidar distance and TTC measurements are minimized here with requiring the bounding box to have certain minimum no of lidar points instead of just > 0.

```
if( currBB->lidarPoints.size()>100 && prevBB->lidarPoints.size()>100 ) // only compute TTC if we have Lidar points
```

FP.6 Performance Evaluation 2
Run several detector / descriptor combinations and look at the differences in TTC estimation. Find out which methods perform best and also include several examples where camera-based TTC estimation is way off. As with Lidar, describe your observations again and also look into potential reasons.

→ Write Up
The above algorithm is run over all the combinations of the matcher , keypoint detector and keypoint descriptor algorithms in the main function.

```
const string DetectorTypes[]=  {"SHITOMASI ","HARRIS   ", "FAST      ", "BRISK    ", "ORB       ", "AKAZE    ", "SIFT     "};
const string DescreptorTypes[]={"BRISK     ","BRIEF    ", "ORB       ", "FREAK    ", "AKAZE    ", "SIFT     "};
const string MatcherTypes[]={"MAT_BF     ","MAT_FLANN    "};
const string SelectorTypes[]={"SEL_NN     ","SEL_KNN    "};
```

The resulting data and analysis images can be found in the /Data folder. The screen output will show following information – selected combination of detector – descriptor – matcher , total processing time , camera TTC , Lidar TTC , difference and abs difference between two TTC.

```
MacherTypes SelectorTypes DetectorType descriptorType FrameNo Processing Time(ms) TTC_Lidar(s) TTC_Camera (s) Difference (s) Abs Difference (s)
```

Unreliable Camera-based TTC

1) The combinations where an CV::exception was returned and we don’t have any keypoints.

DetectorType	descriptorType
SHITOMASI	AKAZE
HARRIS	AKAZE
FAST	AKAZE
BRISK	AKAZE
ORB	AKAZE
SIFT	ORB
SIFT	AKAZE

2) Low number of Keypoint Matches -

As we can see in the data for HARRIS keypoint detector several NAN Camera-based TTC corresponding to a very low or no keypoint matches and/or keypoints. HARRIS keypoint detector may not be suitable for this application

We can see similar results for ORB keypoint detector and say that ORB also may not be suitable keypoint detector.

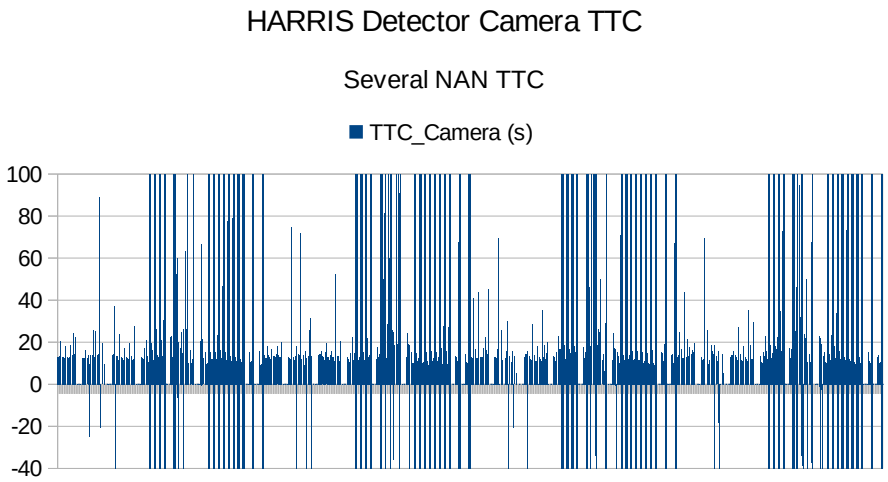

Figure 7

3) Unreliable Camera-based TTC estimates

MacherTypes	SelectorTypes	DetectorType	descriptorType	FrameNo	Processing Time(ms)	TTC_Lidar(s)	TTC_Camera (s)
MAT_BF	SEL_NN	SHITOMASI	BRISK	16	36	9.64521	nan
MAT_BF	SEL_NN	SHITOMASI	FREAK	15	40	8.57106	24.2895
MAT_BF	SEL_NN	BRISK	ORB	6	78	12.8867	26.0184
MAT_FLANN	SEL_NN	BRISK	FREAK	2	97	12.4141	24.0844
MAT_FLANN	SEL_NN	BRISK	FREAK	8	96	13.1207	97.9634
MAT_FLANN	SEL_KNN	SHITOMASI	ORB	19	25	8.90191	21.6134
MAT_FLANN	SEL_KNN	SHITOMASI	FREAK	15	47	8.57106	nan
MAT_FLANN	SEL_KNN	BRISK	FREAK	8	97	13.1207	34.5794
MAT_FLANN	SEL_KNN	BRISK	FREAK	15	99	8.57106	19.3643

- SHITOMASI-BRISK and SHITOMASI-FREAK TTC_Camera estimate of nan is due to very low or no keypoint matches between frames.
- BRISK-FREAK frame no 8 for both rows is way off due to wrong keypoint match making into the bounding box even after filtering. See Figure 8

Figure 8

Some Good Camera-based TTC Observations-

Frames over frame no 50 -
As expected we are seeing very large positive and negative numbers and also INF for camera-based TTC estimates for frames over no 50. The vehicle is stopping or stopped here and we see the same camera image in between frames leading to the distance ratio very close to 1.

FAST Keypoint Detector -

Below Figure 8 shows the FAST keypoint detector with all the combination for descriptor and matchers over the first 50 frames. As we can see the results are very repeatable. The processing time here is quite large may be because we have a very large set of keypoints.
Figure 9 shows a zoom in the area where we have the processing time around < 40 ms. As we can see the FAST BRIEF MAT-BF SEL-KNN combination gives reliable lidar and camera based ttc estimates. The difference being around 2 seconds , but it can be as high as 5 seconds which could be high considering the expected ttc around 12 seconds.

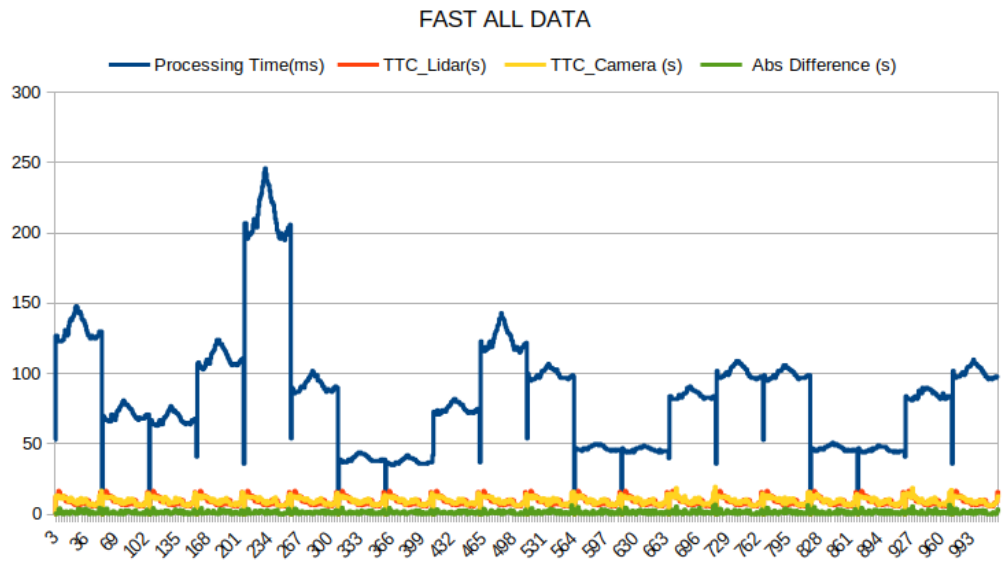

Figure 9

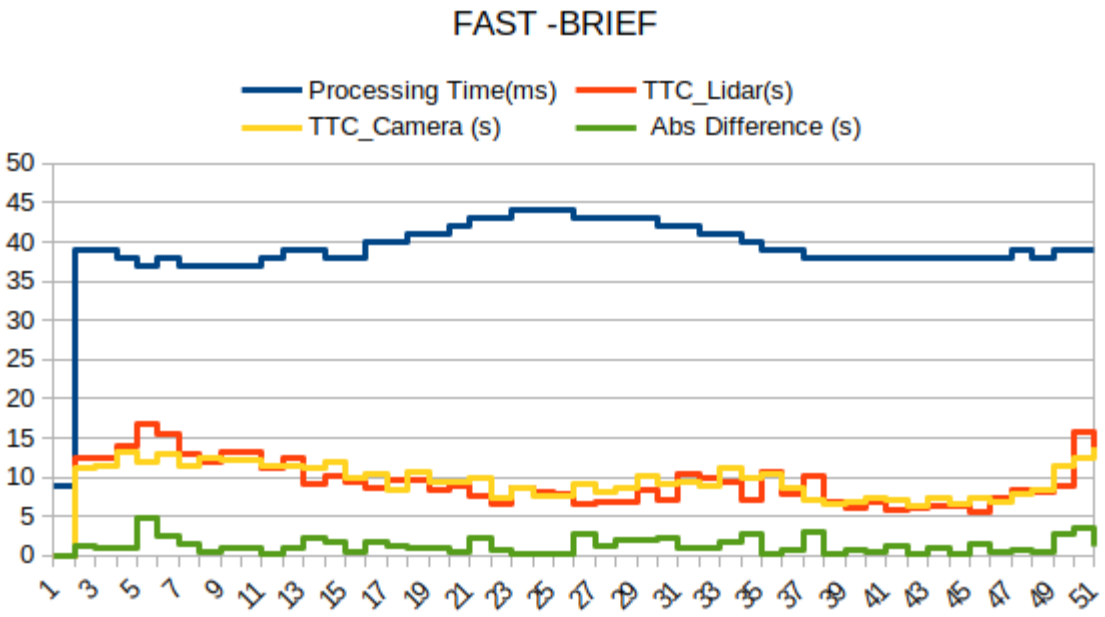

Figure 10

Other Good observations-

FAST-BRIEF (Figure 9) MAT-BF SEL-KNN	<40ms	time diff <5 sec
AKAZE-BRIEF (Figure 10) MAT-BF SEL-KNN	<35 ms	time diff < 5 sec
SHITOMASI-ORB (Figure 11) MAT-BF SEL-KNN	<20ms	time diff <4 sec
SHITOMASI-BRIEF (Figure 12) MAT-BF SEL-KNN	<20ms	time diff <4 sec

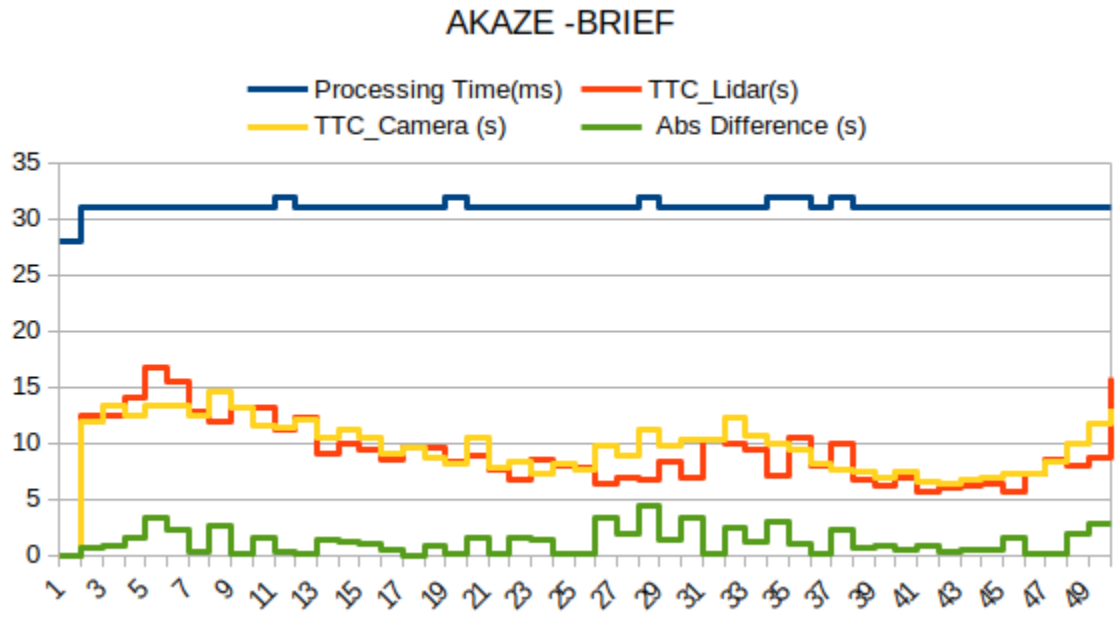

Figure 11

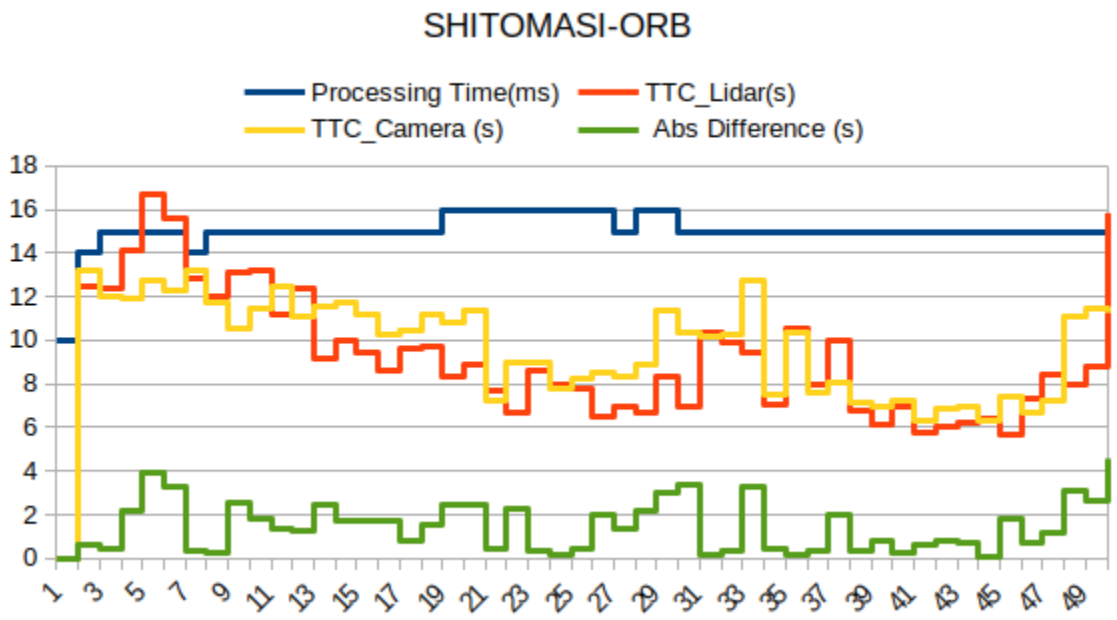

Figure 12

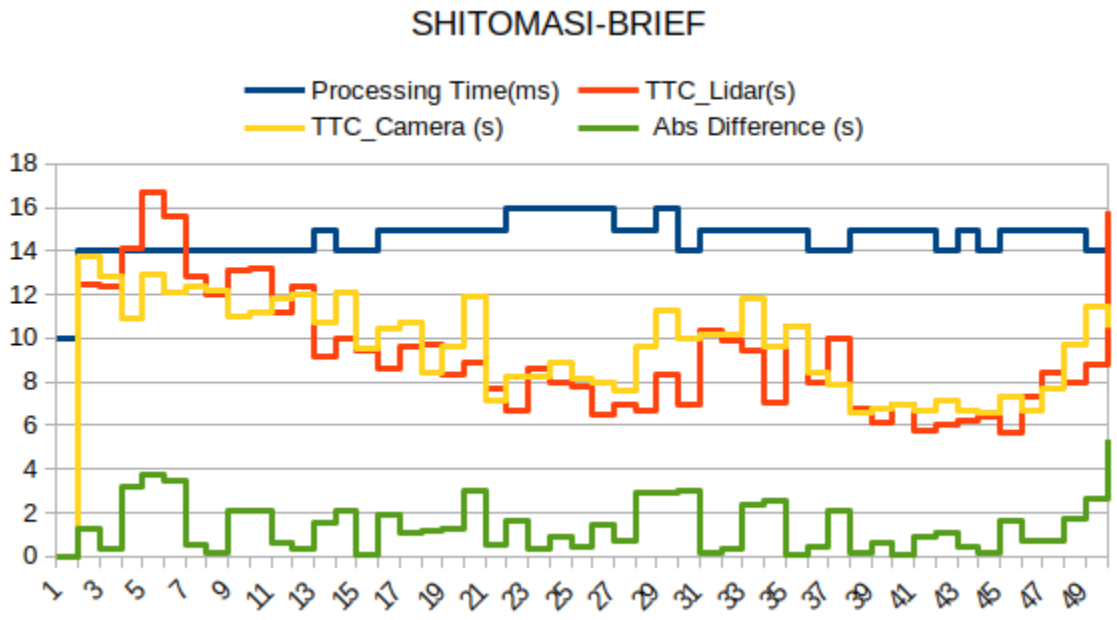

Figure 13

Code Explanation -

```
class helper{
    ObjectDetection2D objDetector_;
    lidar Lidar_;
    double sensorFrameRate;
    int imgStepWidth = 1;
    std::vector<string>cameraFilesNames;
    std::vector<string>LidarFilesNames;

public:
    helper(std::string dataPath, int imgcount);
    int readImage(int imgIndex, boost::circular_buffer<DataFrame> *dataBuffer);

    int detectKeypoints(boost::circular_buffer<DataFrame> *dataBuffer,int detType);
    int descKeypoints_helper(boost::circular_buffer<DataFrame> *dataBuffer,  int
descType);
    int matchDescriptors_helper( boost::circular_buffer<DataFrame>  *dataBuffer,
int descType, int matcherType, int selectorType);
    int matchBoundingBoxes_helper(boost::circular_buffer<DataFrame> *dataBuffer);
    int estimateTTC(boost::circular_buffer<DataFrame> *dataBuffer, bool bVis);
};
```

This class holds the helper functions , which in turn call the functions from ObjectDetection2D and matching2D_student and camFusion_Student files.

helper(std::string dataPath, int imgcount) constructor takes in the path to the image files and the image count. It populates the two vectors with the camera and lidar file names also creates and initializes two objects for ObjectDetection2D and Lidar class.
The ObjectDetection2D class holds all the yolo parameters , functions and process the images for yolo object detection
The Lidar class holds all the parameters and functions for Lidar point projections into camera images.

int **readImage**(int imgIndex, boost::circular_buffer<DataFrame> *dataBuffer); This function reads both camera and lidar files in the circular buffer dataFrame. Also calls the yolo object detection on the camera file . The processed data is stored in dataFrame.

int **detectKeypoints**(boost::circular_buffer<DataFrame> *dataBuffer,int detType);
helper function to call the Keypoint detector functions from matching2D_student.cpp

```
    int descKeypoints_helper(boost::circular_buffer<DataFrame> *dataBuffer,  int
descType);
```

helper function to call the descKeypoints functions from matching2D_student.cpp

```
    int matchDescriptors_helper( boost::circular_buffer<DataFrame>  *dataBuffer,
int descType, int matcherType, int selectorType);
```

helper function to call the Keypoint matcher functions from matching2D_student.cpp

The

```
int matchBoundingBoxes_helper(boost::circular_buffer<DataFrame> *dataBuffer);
int estimateTTC(boost::circular_buffer<DataFrame> *dataBuffer, bool bVis);
```

The first 4 rubric points are addressed in these two functions and the explanation can be found above.

estimateTTC function calls both the cameraTTC and LidarTTC and the values are stored back in the dataFrame for display.