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.

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

Both these rubric points are 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 and put them in respective bounding boxes for both current and previous frame.
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

→ Write Up
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 271 float meanLidarPoint(std::vector<LidarPoint> &lidarPoints , string
windowName)
```

The meanLidarPoint function will calculate the mean and std deviation for all the lidar points and the points which are 1 std deviation away are eliminated as outliers. The resulting data set is averaged to get 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.

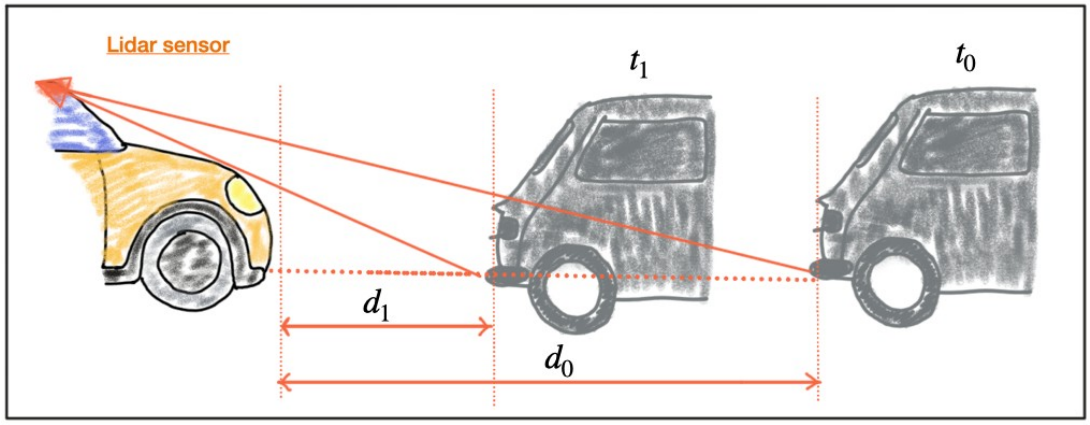

Figure 1

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

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

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

Figure 2

FP.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)
```

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 and the median value of the set is considered robust enough for Camera based TTC estimation. Following Figure 3 and Figure 4 camera based motion model is used to calculate the TTC.

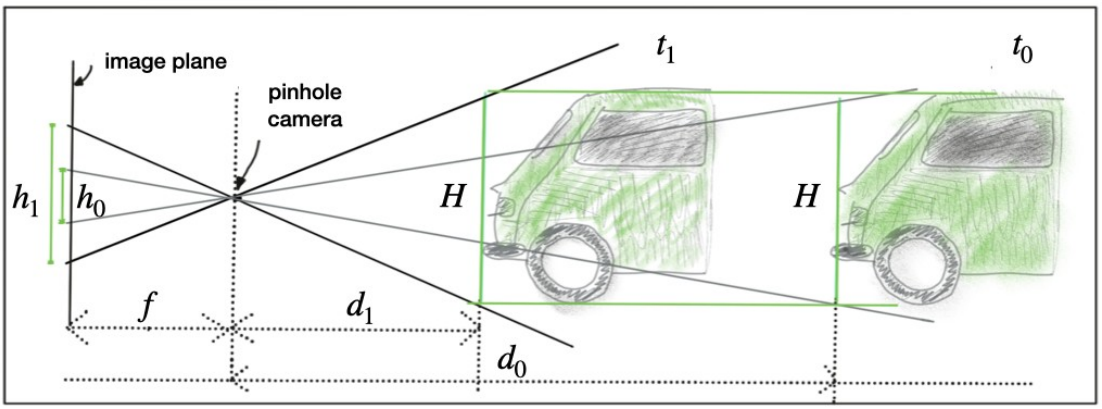

Figure 3

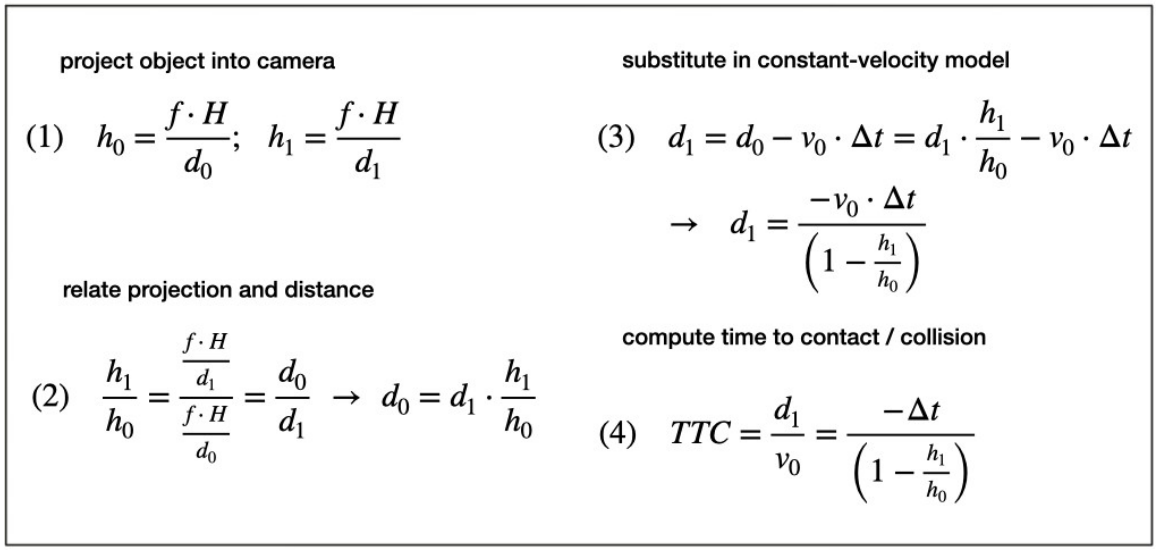

Figure 4

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.
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)
```

Analysis -

FAST Keypoint Detector -

Below Figure 5 shows the FAST keypoint detector will 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 6 shows a zoom in the area where we have the processing time around < 50 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 7.5 seconds which could be high considering the expected ttc around 12 seconds.

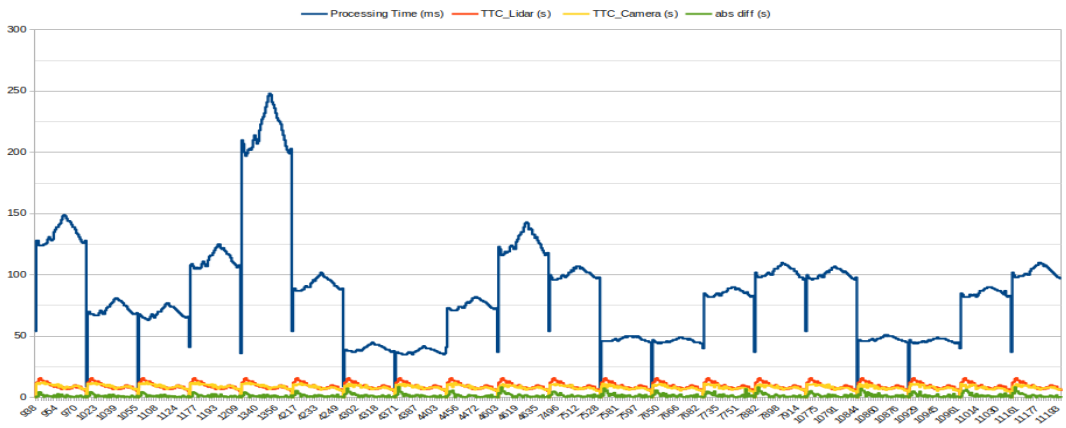

Figure 5

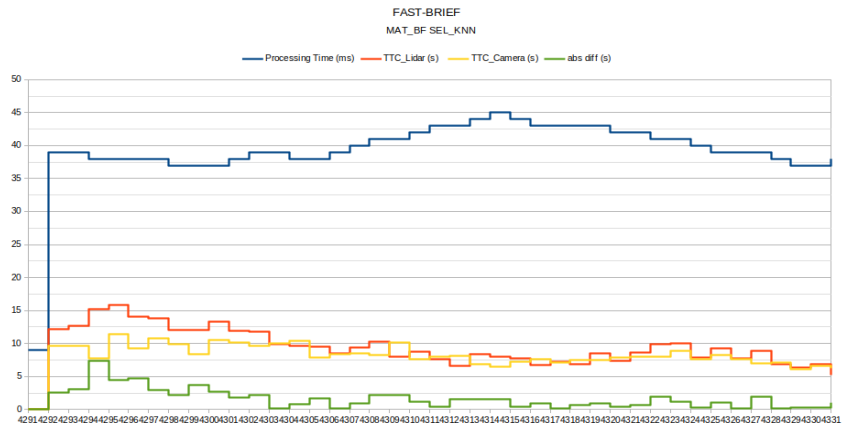

Figure 6

Other Good observations-

FAST-BRIEF (Figure 6)	<50ms	Max time diff 7.5 sec
AKAZE-BRIEF (Figure 7)	<40 ms	Max time diff 5 sec
SHITOMASI-ORB (Figure 8)	<20ms	Max time diff 5 sec
SHITOMASI-BRISK (Figure 9)	<20ms	Max time diff 5 sec

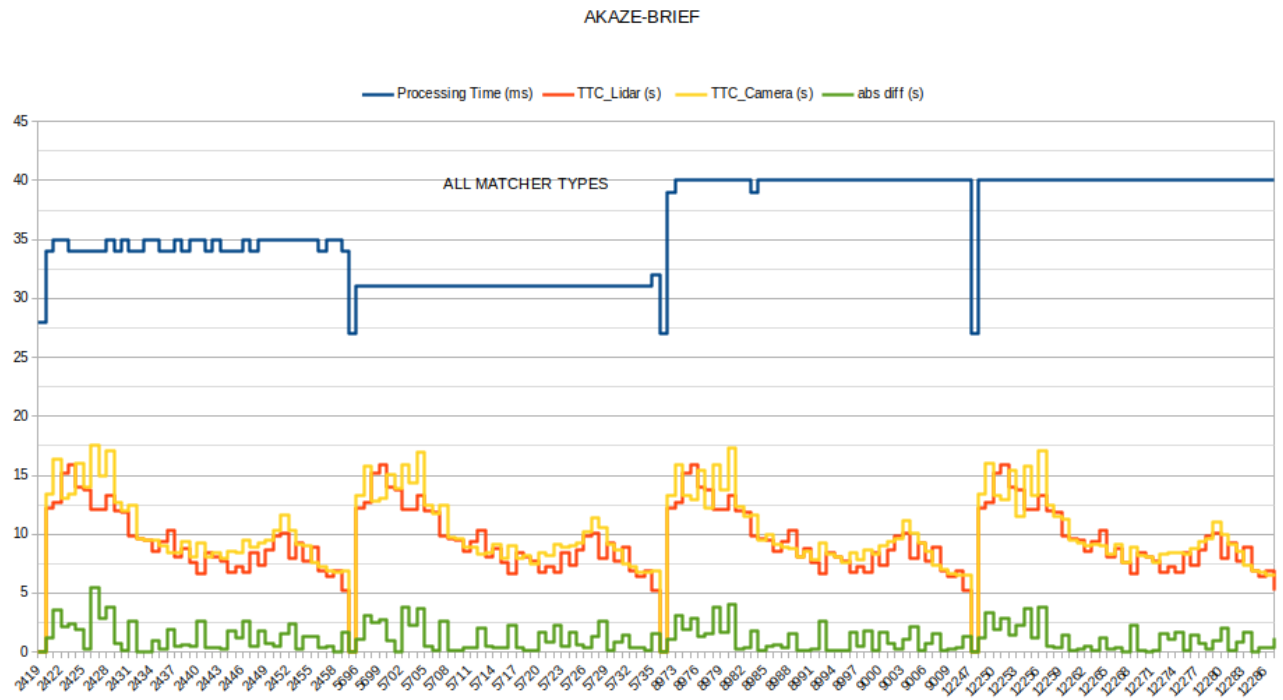

Figure 7

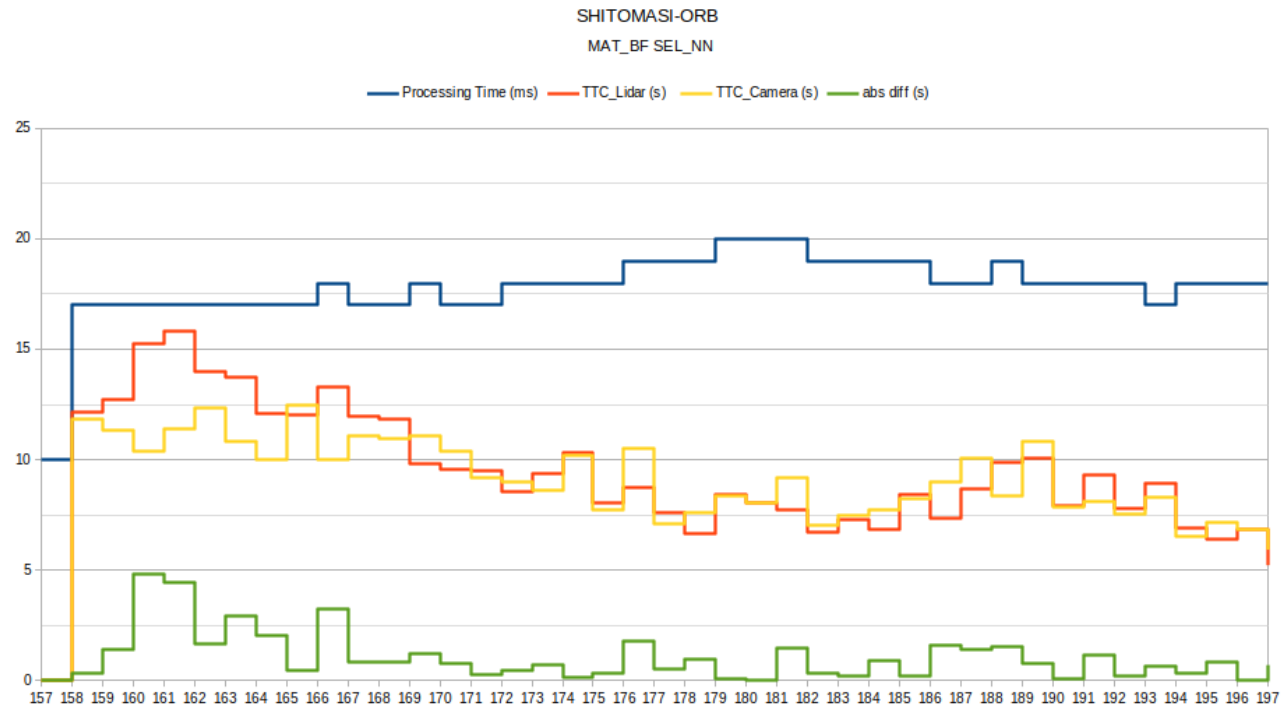

Figure 8

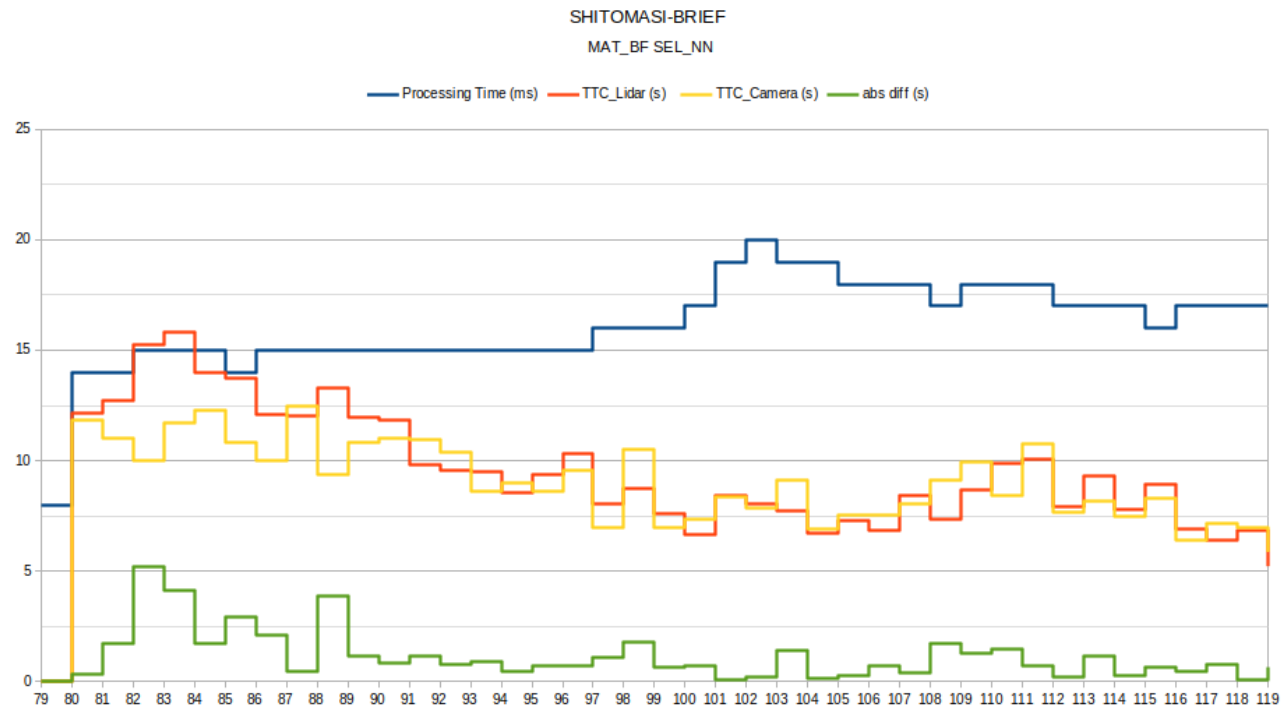

Figure 9

Unreliable Estimates -

Figure 10 11 shows the datapoints where camera-based TTC , Lidar-based TTC or both estimates are unreliable. As we can see with Fig 10 HARRIS_ALL data there are several NAN datapoints corresponding to no keypoint matches. We conclude that HARRIS keypoint detector is algorithm not suitable for this application.

Also supporting data is available in File Ref – Unreliable_TTC.html show that the both Lidar-based TTC and Camera-based TTC fail for the frames nos > 50. Figure 12 shows those failures. A closer analysis is presented in File Ref – Unreliable_TTC.html. Looking at it we can see that the either the vehicle is not moving or it is too close for lidar and camera to detect the deltas. In this case the motion model is very unreliable , we don't know if the velocity estimates are accurate ? Are we already in collision? We may need to look into other sensor inputs to know for sure.

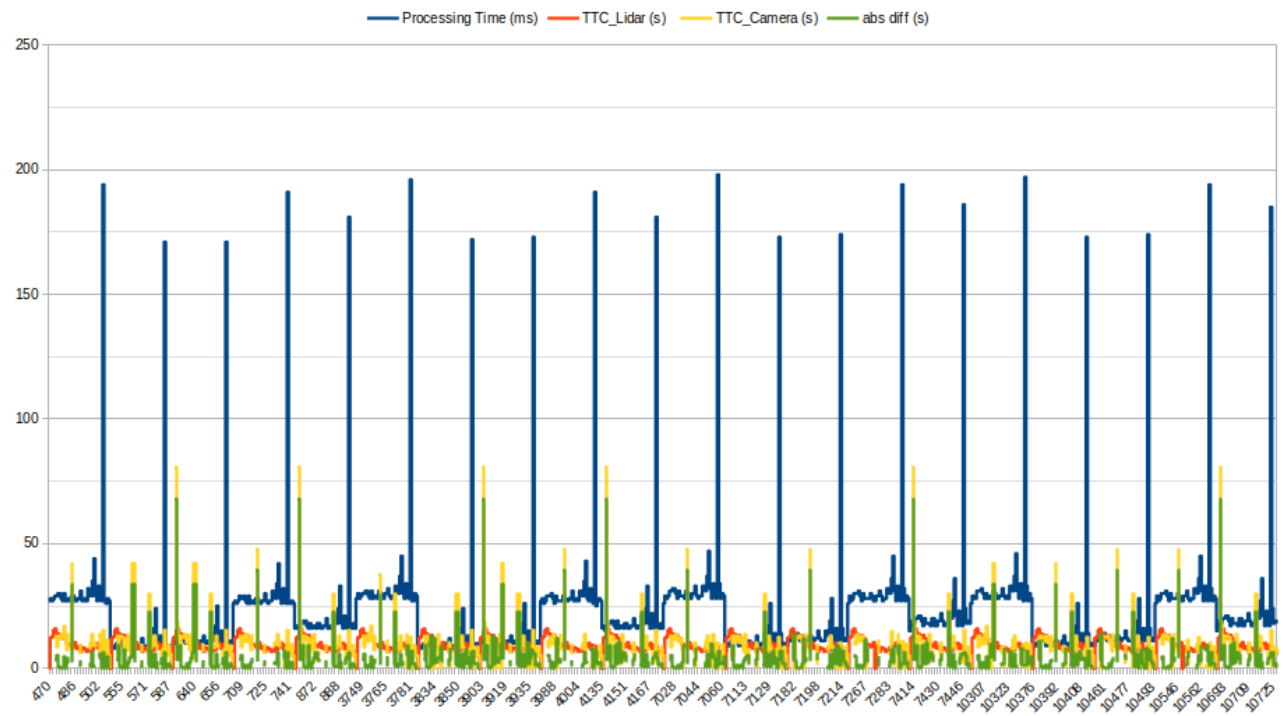

Figure 10 HARRIS_ALL_DATA

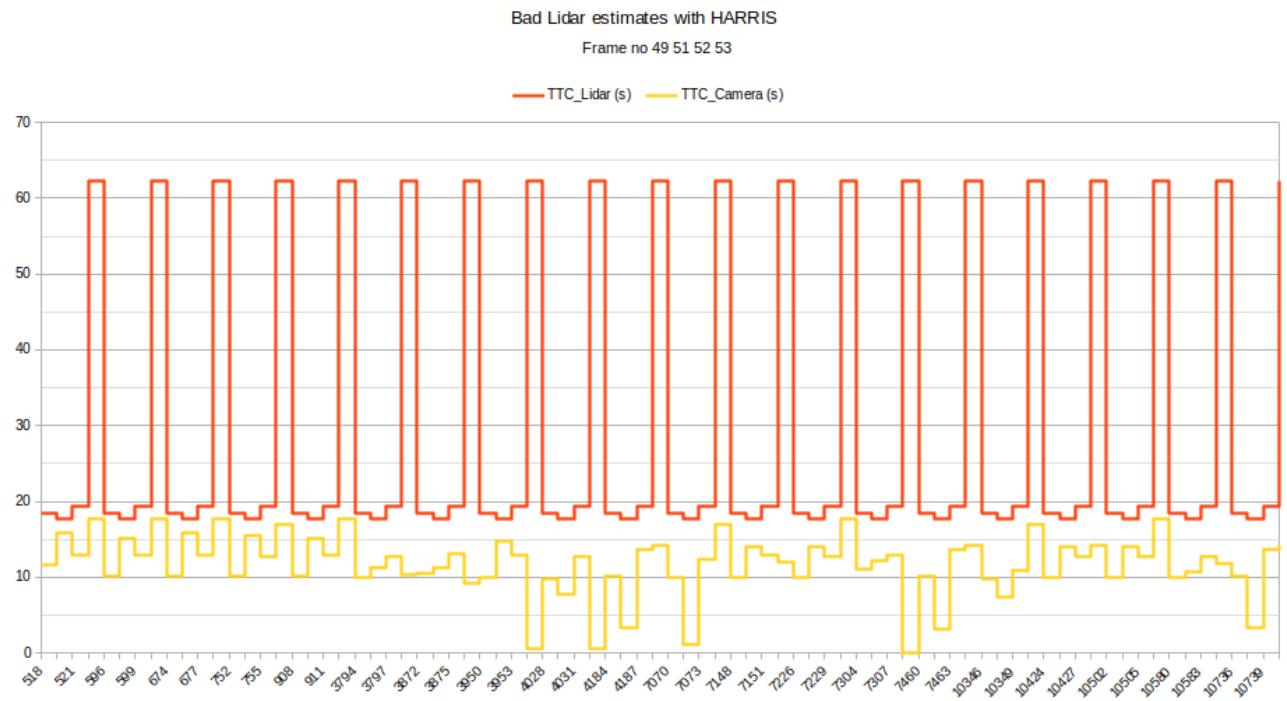

Figure 11

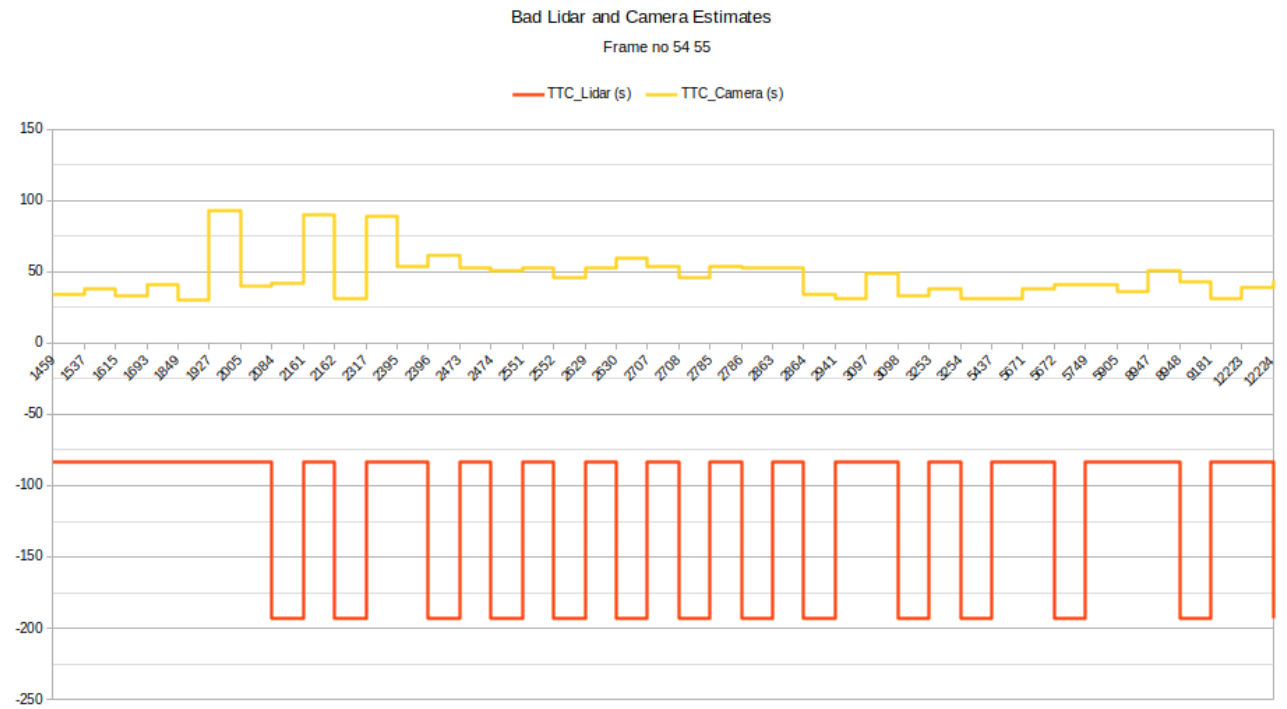

Figure 12

File Ref – LidarAnalysis.html for frames no > 50 , where the distance estimates are around 4.5 , a

closer analysis is done to look into why the Lidar TTC based estimates are way off. Looking at this both the xmin or xavg readings between subsequent frames become too close and the estimates become unreliable.

File Ref – Unreliable_TTC.html

File Ref - Closer snap shot of some or the algorithms and frames where both TTC and Lidar estimates are unreliable .

File Ref - data_new.csv the original data

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.