**Project3: Track an Object in 3D Space**

**Write up** 1st submit:October 20th, Kenta Kumazaki

**1. Background**

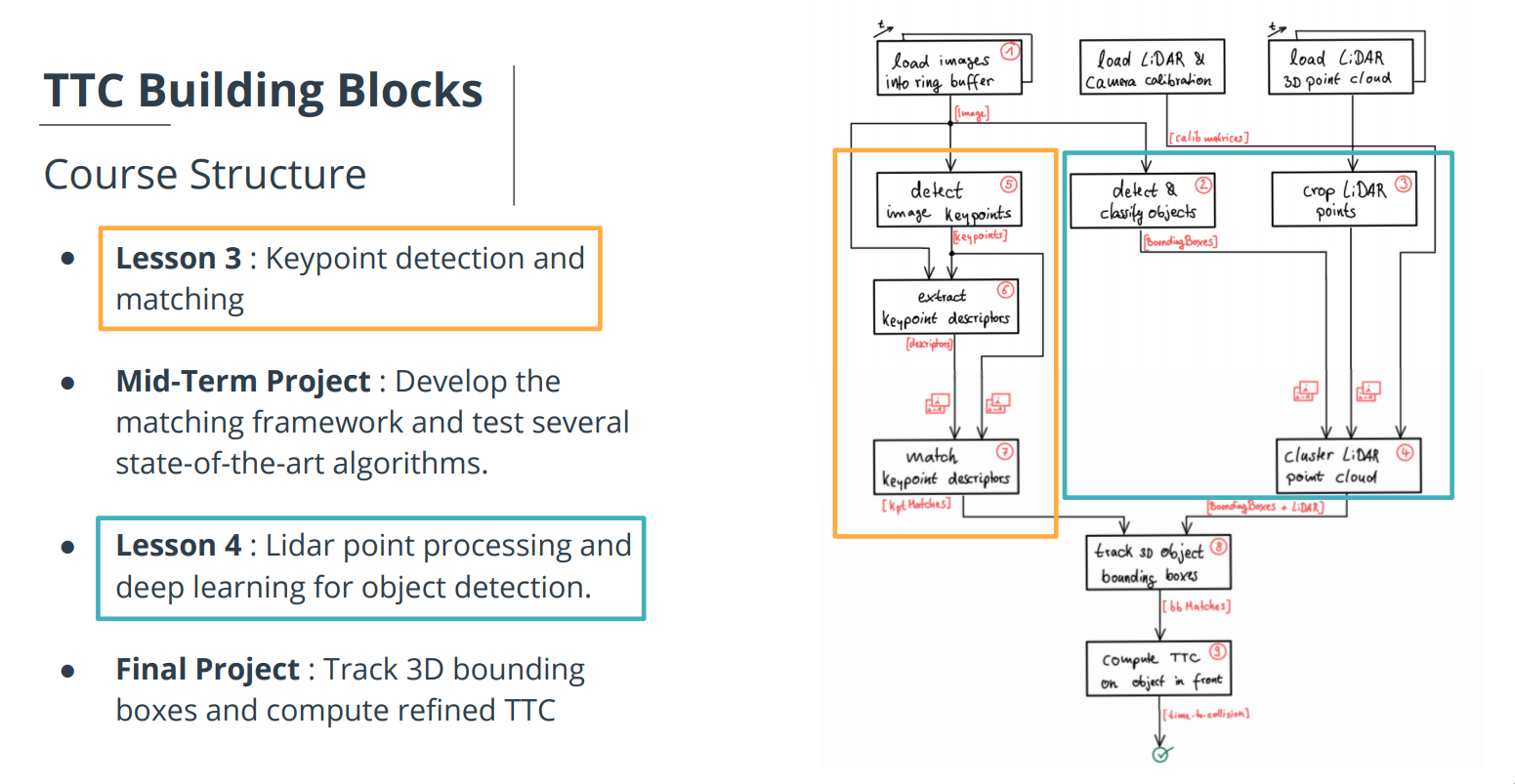
By completing all the lessons, I learned keypoint detectors, descriptors, and methods to match them between successive images. Also, I know how to detect objects in an image using the YOLO deep-learning framework.

And finally, I know how to associate regions in a camera image with Lidar points in 3D space.

What I have learned in the lessons are contained in the following repository.

https://github.com/kkumazaki/Sensor-Fusion\_Camera\_Lessons.git

The program schematic shows what I already have accomplished and what's still missing.



**2. Goal**

In this final project, I implemented the missing parts in the schematic. To do this, I completed four major tasks:

1. I developed a way to match 3D objects over time by using keypoint correspondences.

(2) I computed the TTC based on Lidar measurements.

(3) I proceeded to do the same using the camera, which requires to first associate keypoint matches to regions of  
 interest and then to compute the TTC based on those matches.

(4) I conducted various tests with the framework. My goal is to identify the most suitable detector/descriptor  
combination for TTC estimation and also to search for problems that can lead to faulty measurements by the camera or Lidar sensor.

\*: In the last course of this Nanodegree, I will learn about the Kalman filter, which is a great way to combine the two independent TTC measurements into an improved version which is much more reliable than a single sensor alone can be.

**3. Submission**

**(1) GitHub**

<https://github.com/kkumazaki/Sensor-Fusion_Project3_Track-an-Object-in-3D-Space.git>

**(2) Directory**

I cloned the basic repository from Udacity https://github.com/udacity/SFND\_3D\_Object\_Tracking.git

and added/modified the following files.

* **Writeup\_of\_project3.pdf**: This file
* **README.md**: Read me file of this repository
* **src**
  + **FinalProject\_Camera.cpp**: Main script to set the initial conditions and run the functions.
  + **camFusion\_Student.cpp**: Script used to create the functions of Track 3D Object Bounding Boxes and Compute TTC on Object in front..
  + **matching2D\_Student.cpp**: Script made in Project 2. (detectors, descriptors and matchers)
* **result**
  + **Project3\_result.xlsx**: The resulting list of calculating TTC.
  + **detector\_\*\*\*\_descriptor\_\*\*\*.txt**: The result of calculation with each combination of detectors/descriptors.

**4. Reflection**

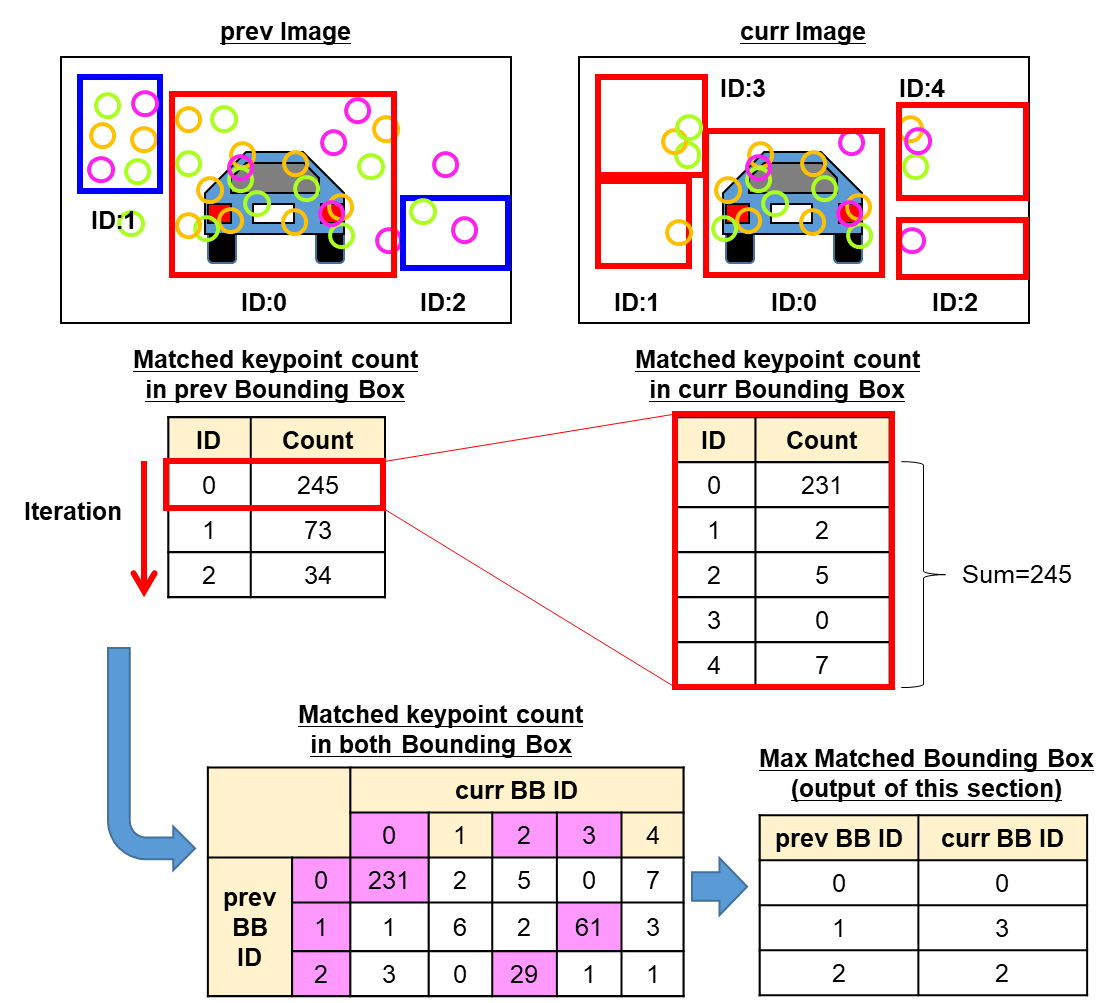
**(1) Match 3D Objects**

**Task FP.1**

In this task, I implemented 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.

The task is complete once the code is functional and returns the specified output, where each bounding box is assigned the match candidate with the highest number of occurrences.

The output image of a simple example is as following:

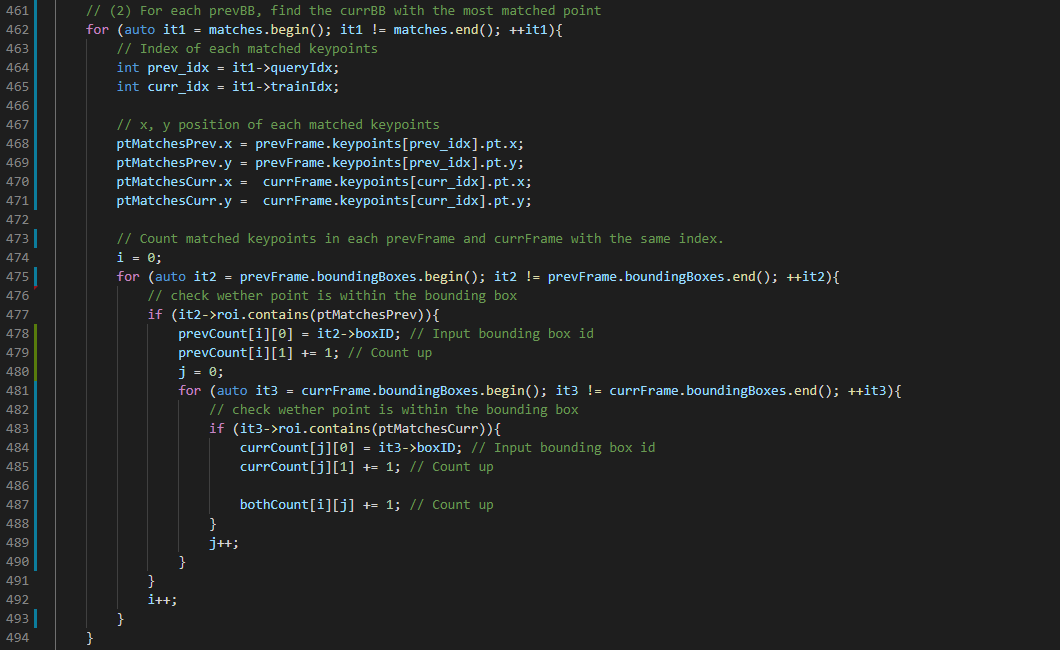


The code of the method "matchBoundingBoxes" is shown below. (camFusion\_Student.cpp)



The double for loops create “bothCount[ i ][ j ]”, which is the matched keypoint count in both bounding boxes. **(A)**

(other counting matrices: prevCount and currCount are used for only debug)

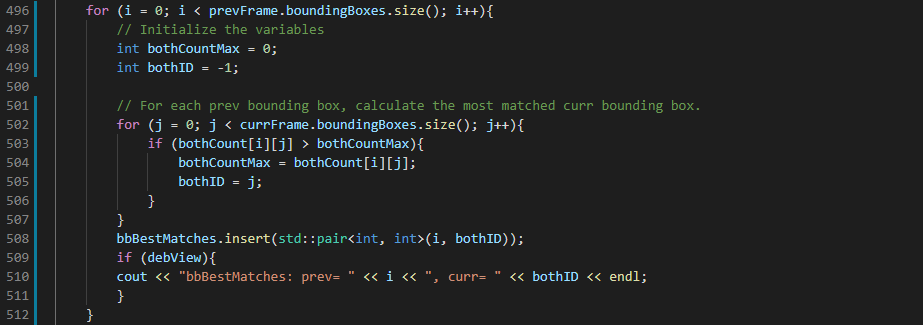


**A**

**C**

**A**

As written in the output image in the previous page, I calculated the max matched curr bounding box for each prev bounding box and insert to the output map “bbBestMatches”.



**A**

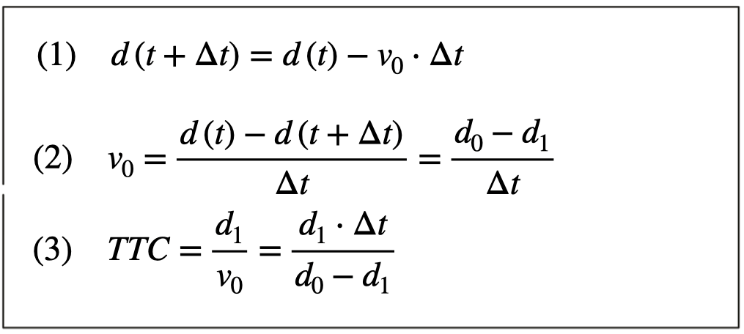
The real output of this method is shown below.



**(2) Compute Lidar-based TTC**

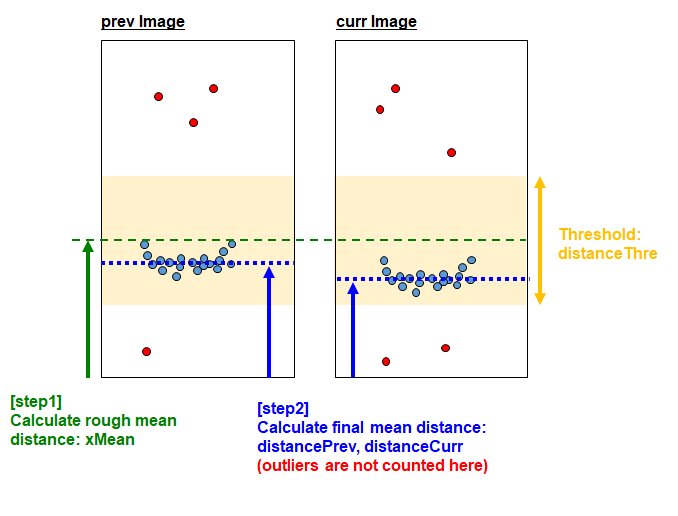
**Task FP.2 : Compute Lidar-based TTC**

In this part of the final project, my task is to compute the time-to-collision for all matched 3D objects based on Lidar measurements alone. I referred to the "Lesson 3: Engineering a Collision Detection System" of this course to revisit the theory behind TTC estimation show as below.



Also, I implemented the estimation in a way that makes it robust against outliers which might be way too close and thus lead to faulty estimates of the TTC. Then I return my TCC to the main function at the end of the method “computeTTCLidar”. The task is complete once the code is functional and returns the specified output. Also, the code is able to deal with outlier Lidar points in a statistically robust way to avoid severe estimation errors.

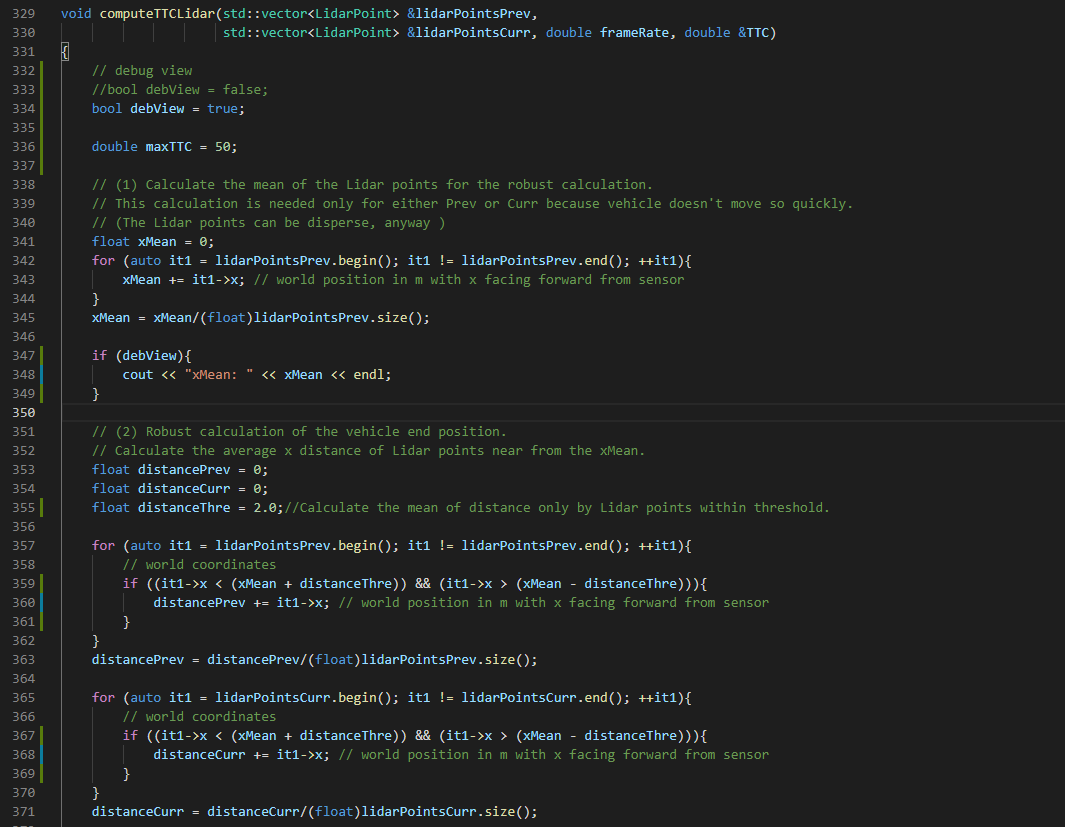
The output image of my code is shown below:



My code is show below. At first, I calculate the rough mean distance of the prev Lidar Points: xMean. **(A)**

I may be able to calculate rough mean for both prev points and curr points. However the real distance doesn’t move so much with quite a small time step, so I only do it for prev points to save calculation time.

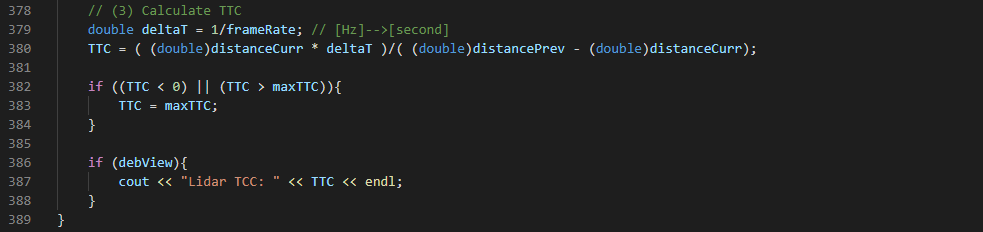
By using rough mean and threshold, I remove the outliers during my final mean distance calculation. **(B)**



**B**

**A**

Finally, I calculate the Lidar TTC according to the equation.



**(3) Compute Camera-based TTC**

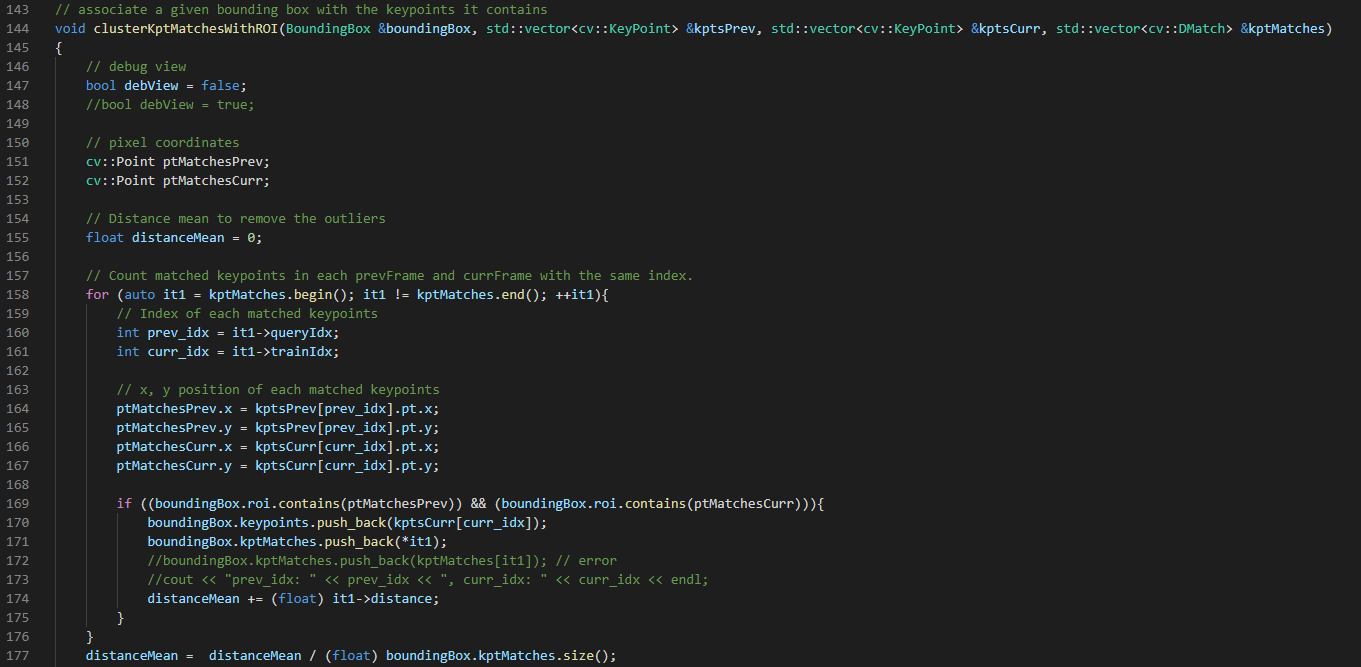
**Task FP.3 : Associate Keypoint Correspondences with Bounding Boxes**

Before a TTC estimation of Camera, I need to find all keypoint matches that belong to each 3D object.

I can do this by simply checking whether the corresponding keypoints are within the region of interest in the camera image. All matches which satisfy this condition should be added to a vector.

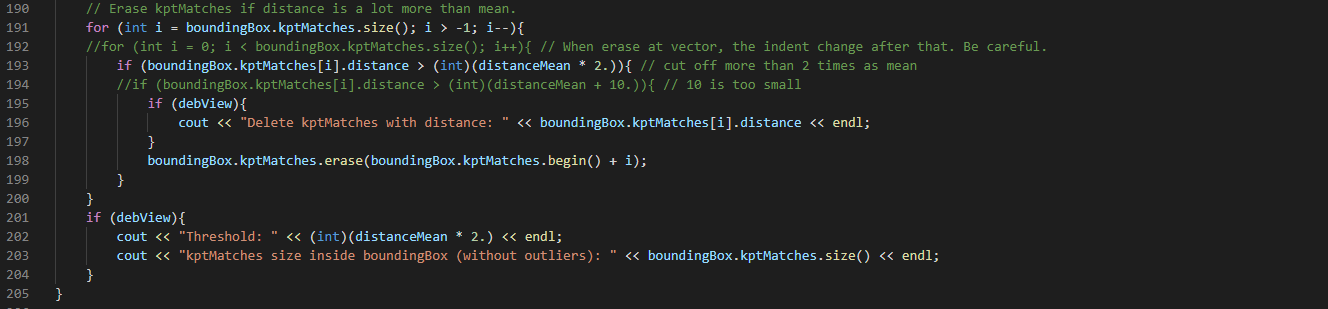
There are outliers among my matches, so I should calculate a robust mean of all the euclidean distances between keypoint matches and then remove those that are too far away from the mean.

My code is shown below. First of all, I calculate the “distanceMean”. **(A)**



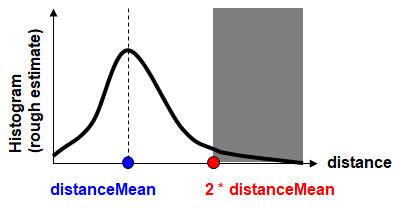
**A**

I erase the keypoint matches whose distance is greater than the twice amount of “distanceMean”. **(B)**



**B**

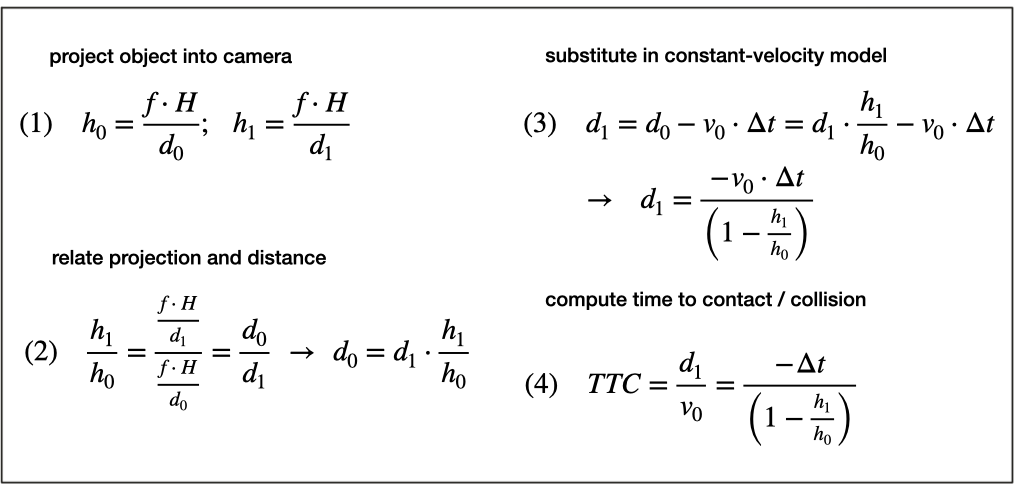
The rough output image is as below.



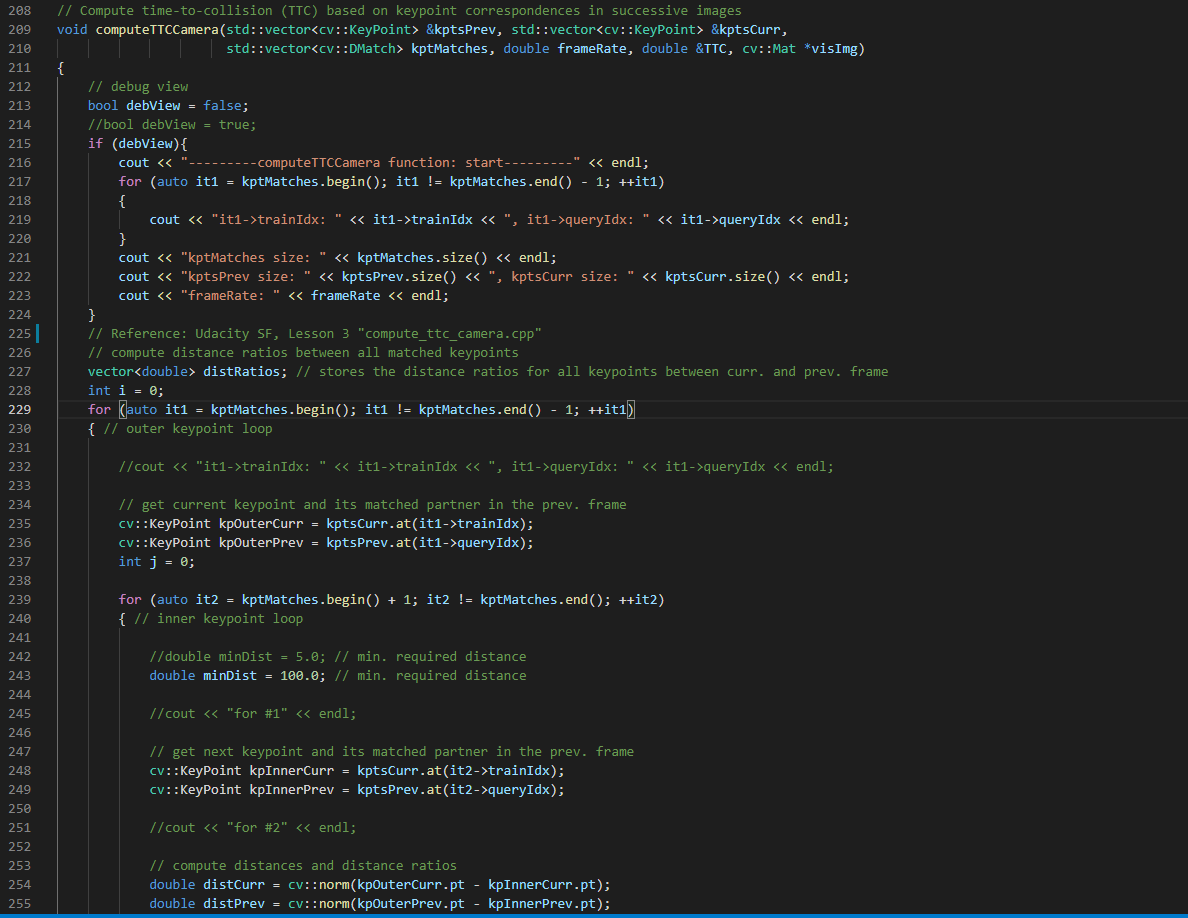
**Task FP.4 : Compute Camera-based TTC**

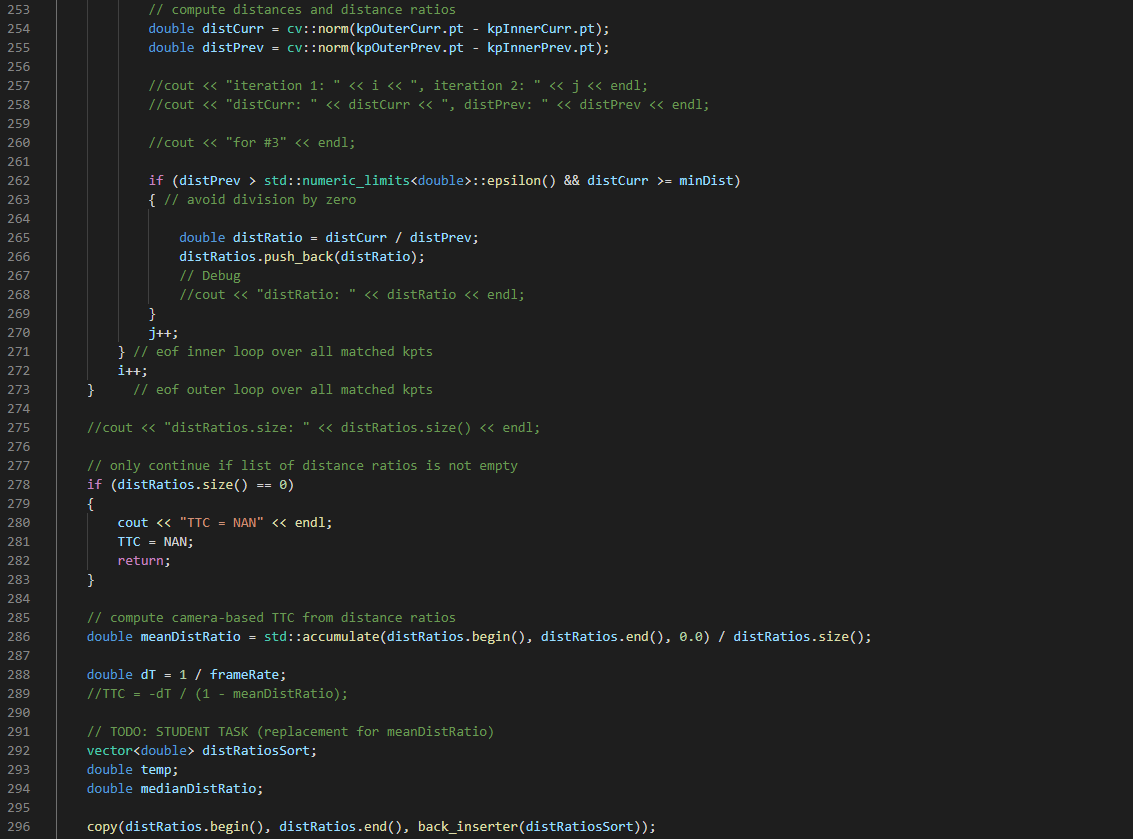
Once keypoint matches have been added to the bounding boxes, the next step is to compute the TTC estimate. I refer Lesson 3 "compute\_ttc\_camera.cpp" and use the code sample there as a starting point for this task here.

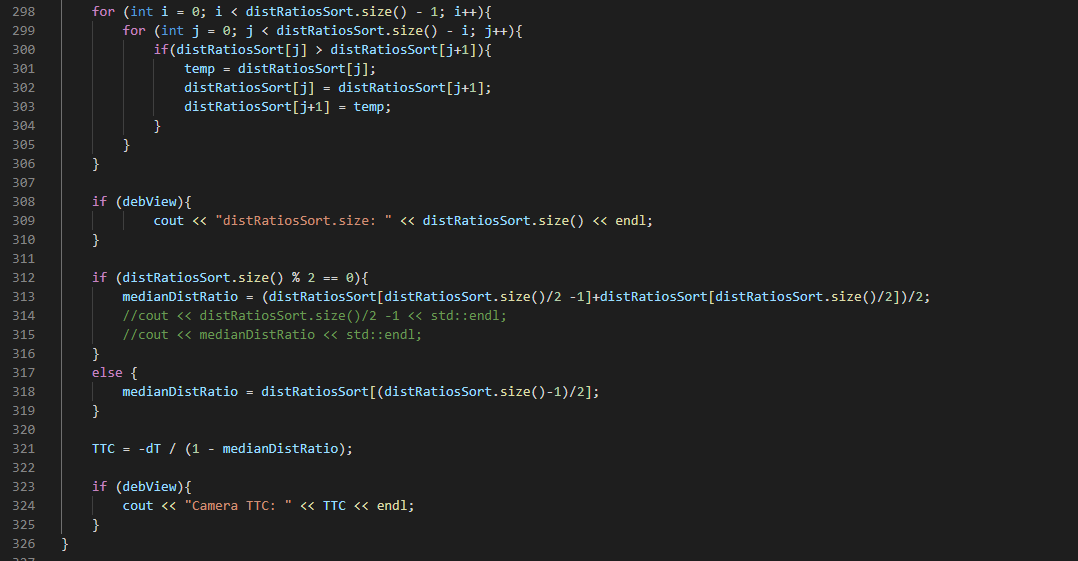
Camera TTC is shown below.



The following is my code.







**(4) Performance Evaluation**

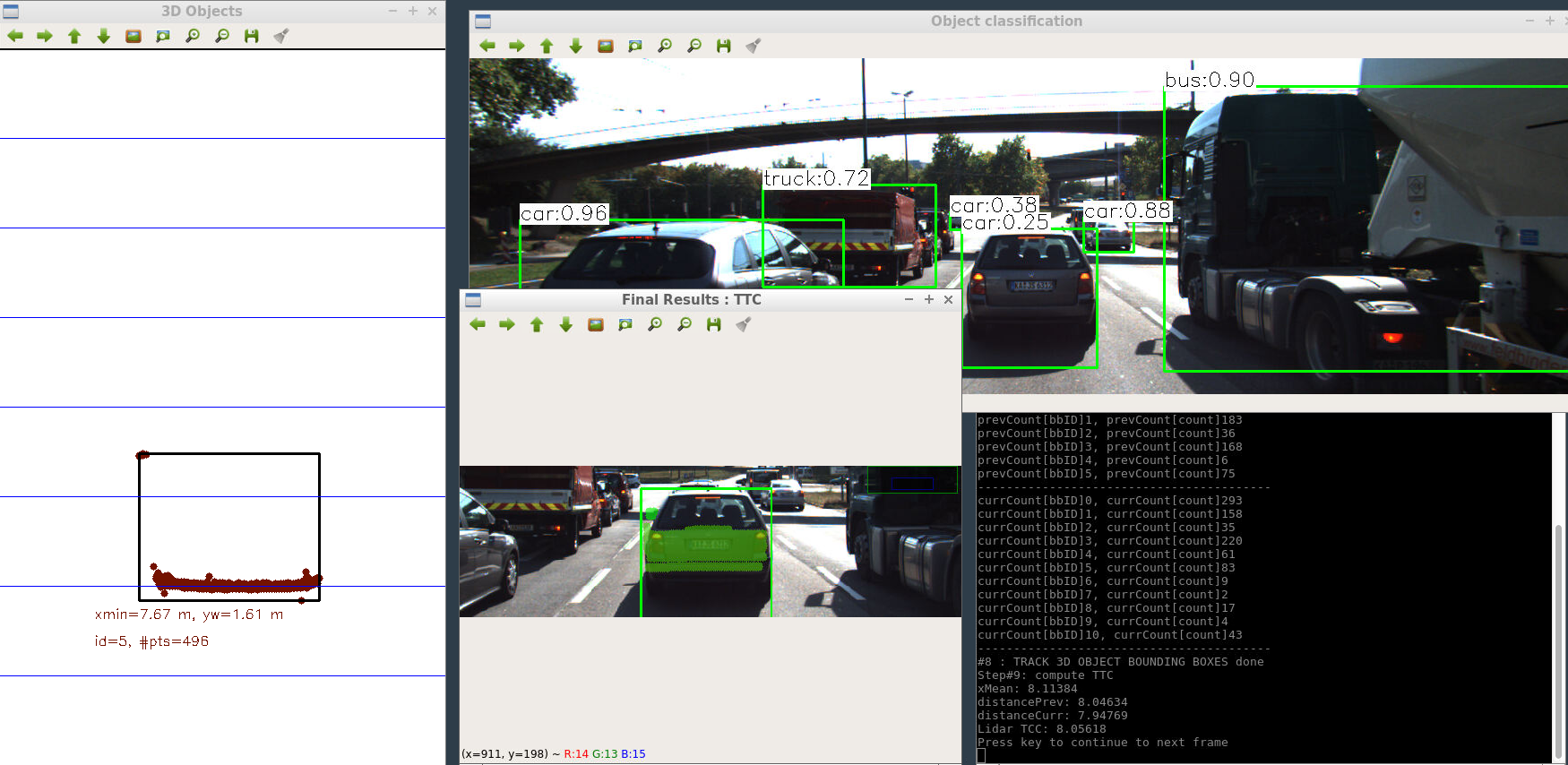
**Task FP.5 : Performance Evaluation 1**

This exercise is about conducting tests with the final project code, especially with regard to the Lidar part.

Look for several examples where I have the impression that the Lidar-based TTC estimate is way off.

Once I have found those, describe my observations and provide a sound argumentation why I think this happened.

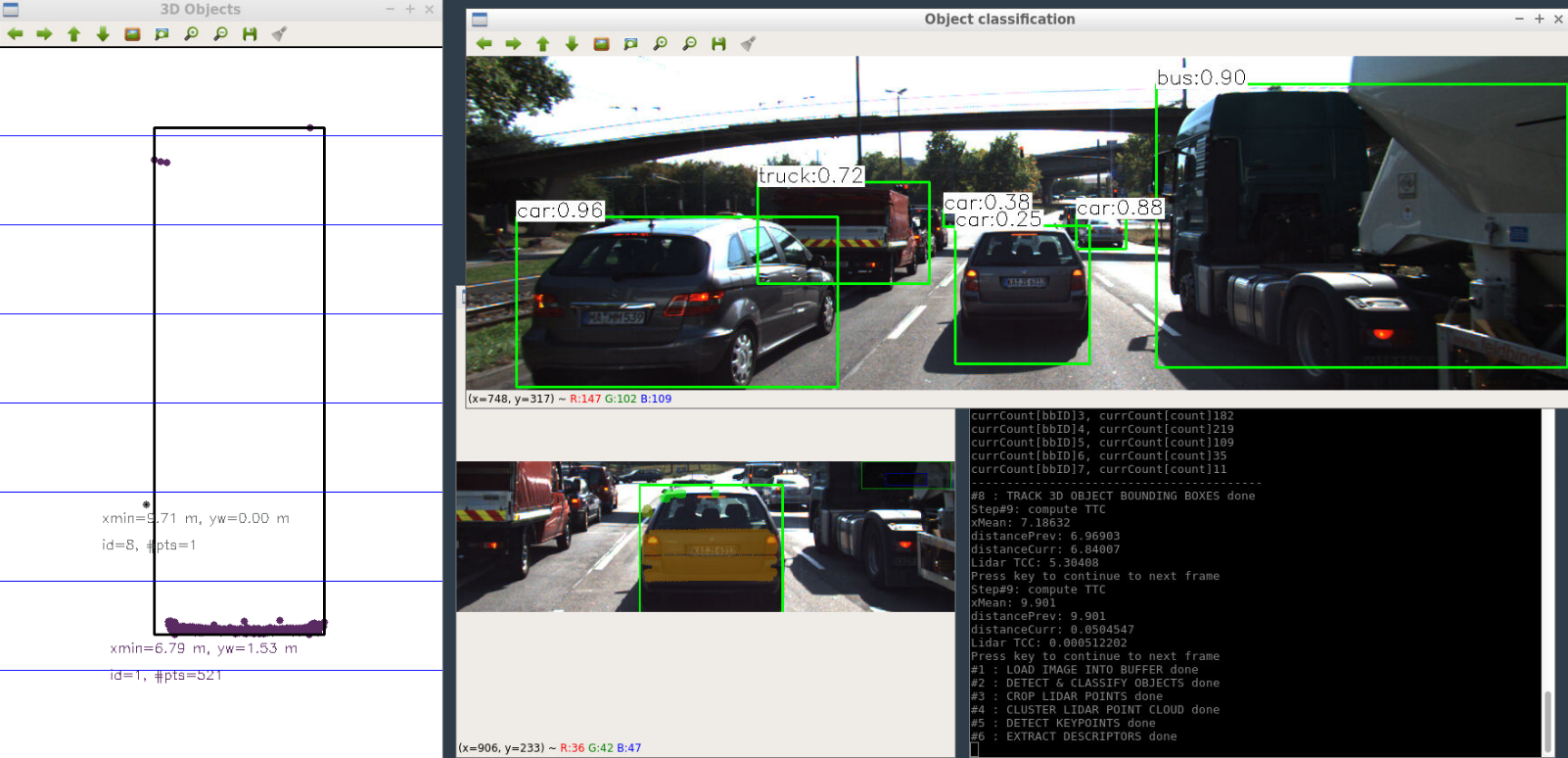
Basically there’s 1 matched object appear and Lidar TTC is calculated stably as below.



However in imageID=17, there appear 2 matched objects. **(A)**

With that unnecessary matched object, Lidar TTC becomes zero and it’s not correct. **(B)**

In this kind of situation, I can reject the far different result than previous result and take the other one. **(C)**



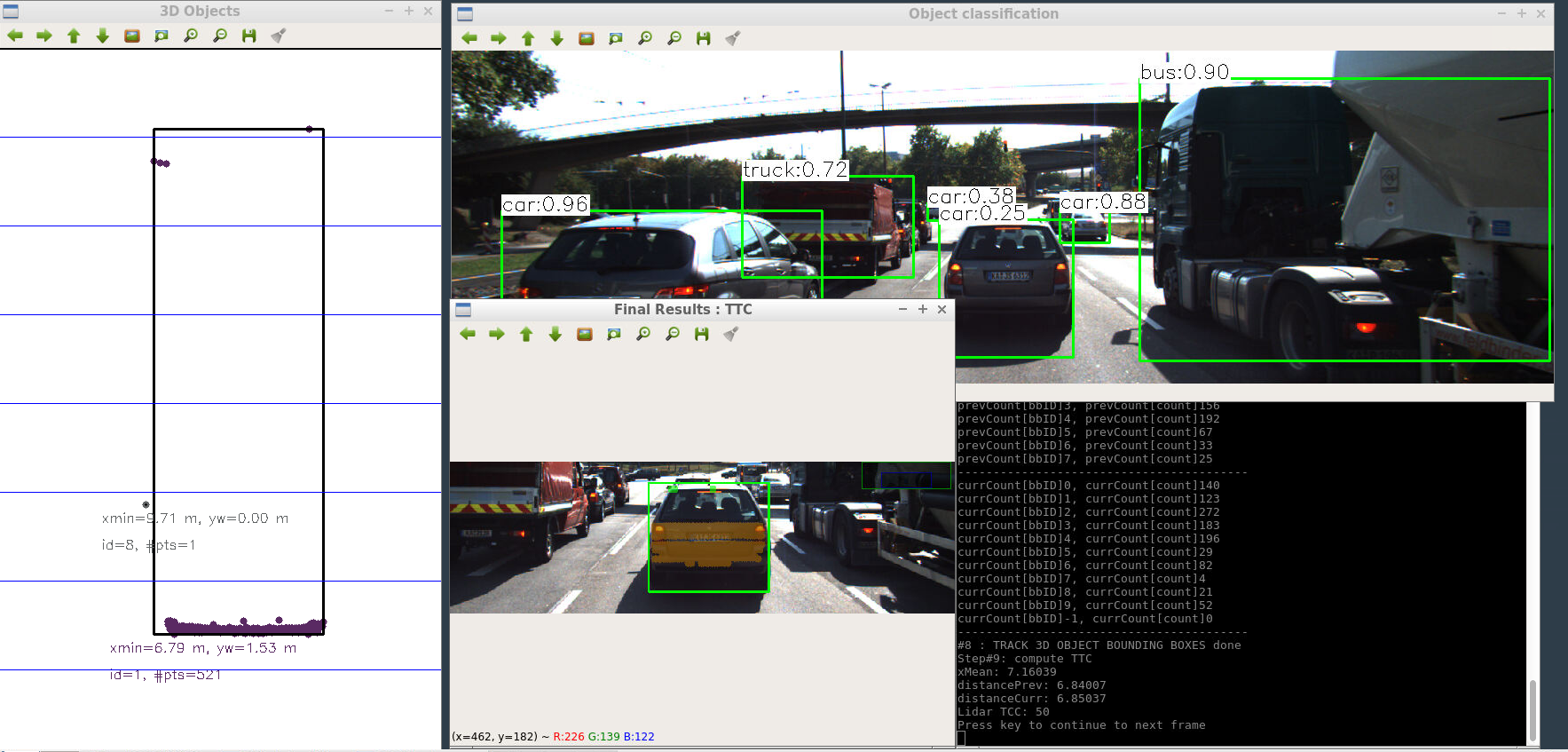
**C**

**B**

**A**

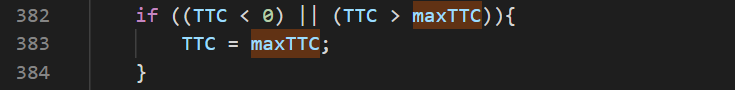
In imageID=18, the distancePrev is smaller than distanceCurr. **(A)**

In this case TTC would be calculated as minus, but minus time doesn’t make sense.

****

**A**

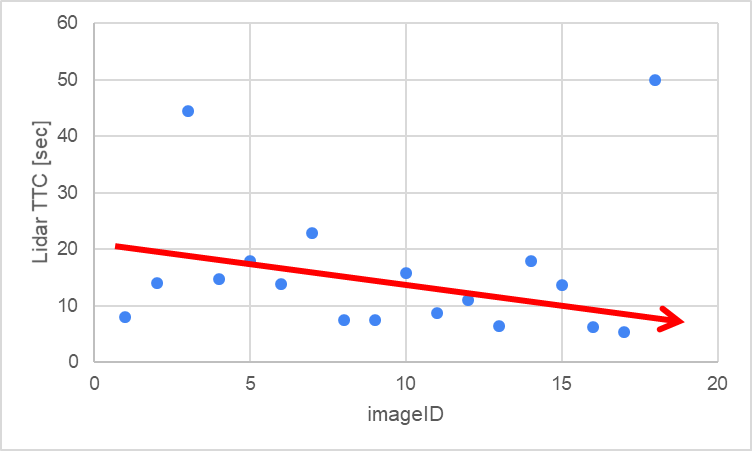
So I made a moderately large time as a threshold (maxTTC = 50), and I added the guard for minus TTC or too large TTC as below. That’s why the resulting TTC is “50” in this scene. **(A)**

****

The resulting Lidar TTC is shown below.

There are some noise and outliers as described above, but overall Lidar TTC keeps decreasing

as the preceding vehicle gets closer to ego vehicle.

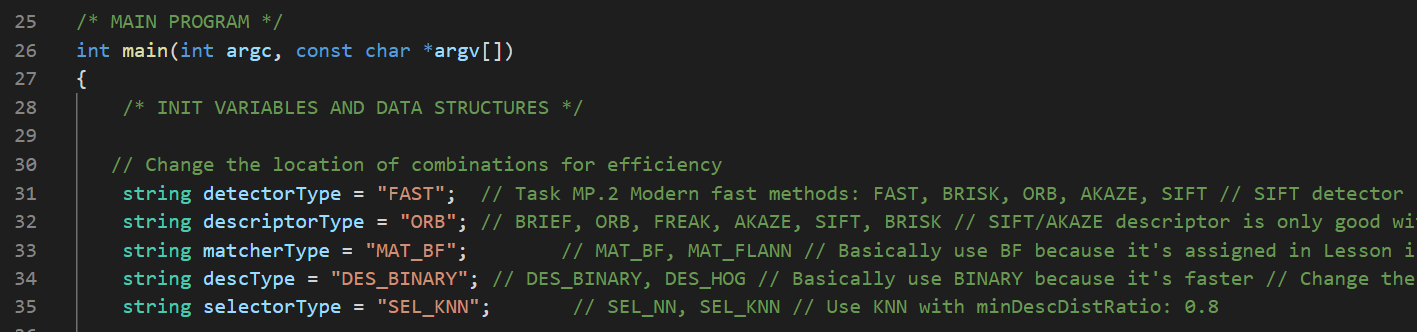


**Task FP.6 : Performance Evaluation 2**

**(1)Basic result and analysis of Camera TTC**

This last exercise is about running the different detector / descriptor combinations and looking 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, I describe my observations again and also look into potential reasons. This is the last task in the final project.

As the basic setup, I’ve been using “detector: FAST” and “descriptor: ORB”, which was the best combination in my Project 2: Camera Based 2D Feature Tracking.



First of all, I show the resulting Camera TTC as below.

Except for one outlier, there’s less variance than Lidar TTC.

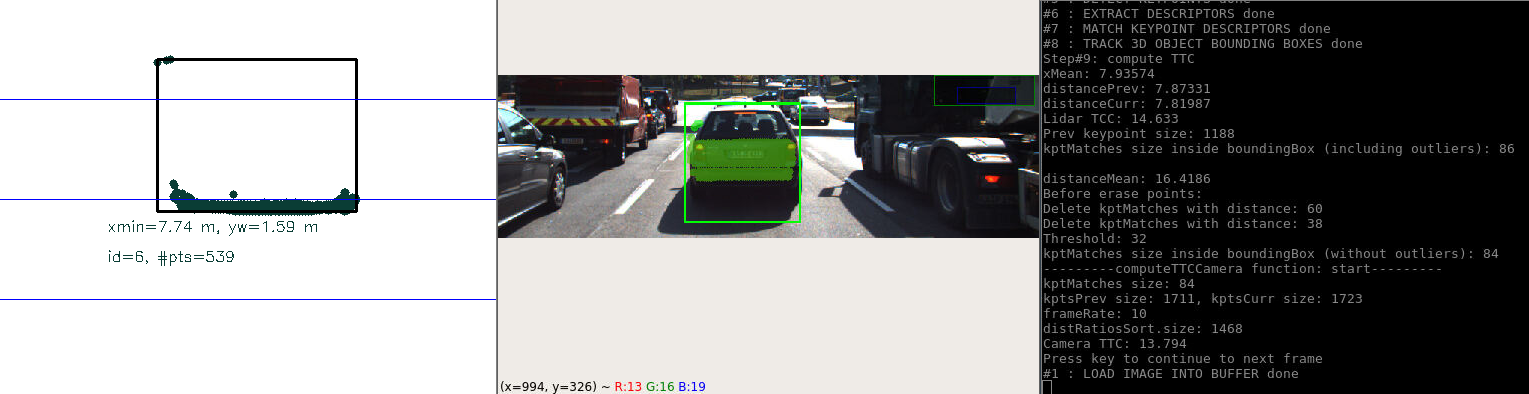


**imageID=5**

**outlier**

I will explain the details in the next page.

The following imageID=4 is the right before outlier occurs. Bounding box is covering only the preceding vehicle.

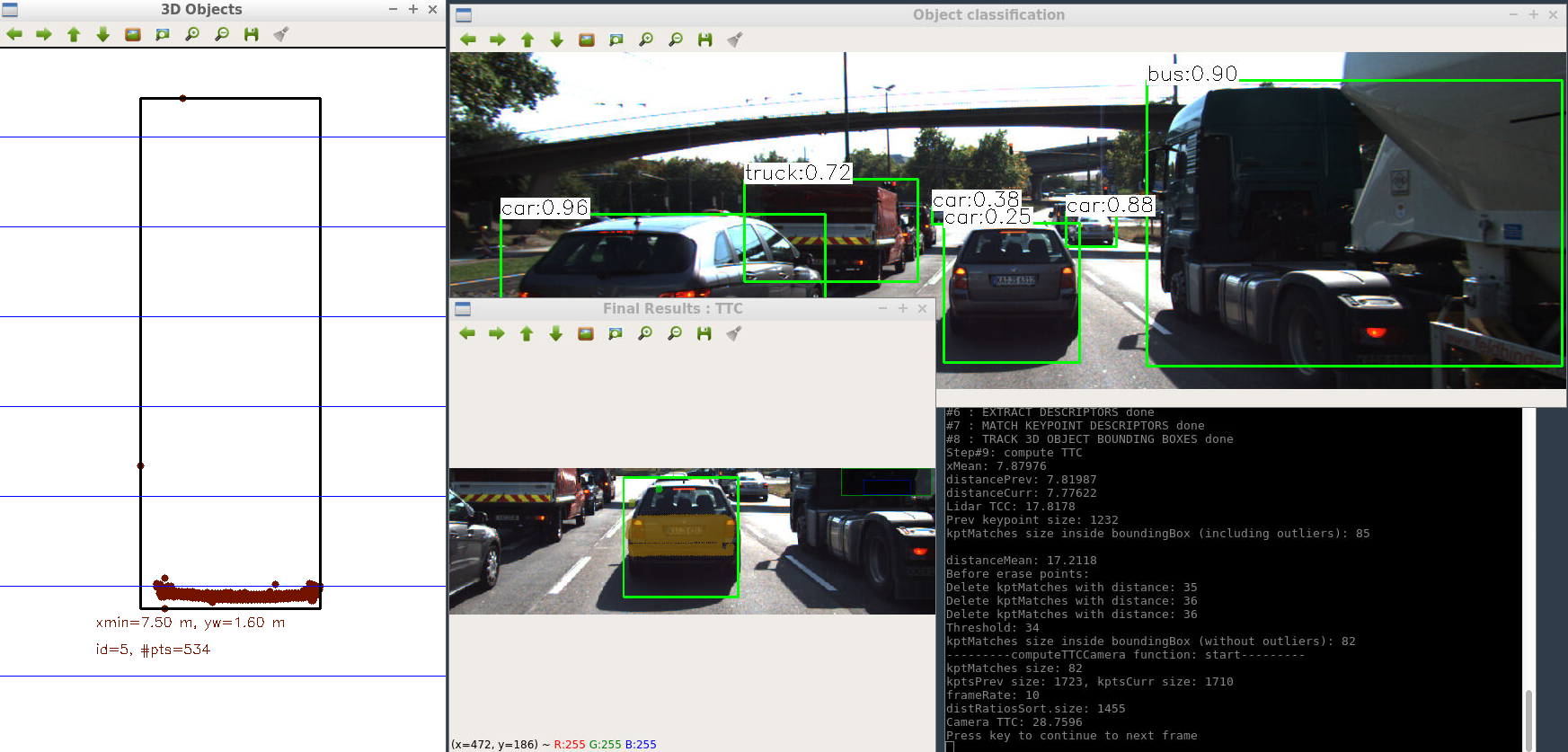


imageID=4

CameraTTC = 13.794

The following imageID=5 is the exact timing of the outlier occurs. The difference between imageID=4 is that the Lidar detects the points of the vehicle in front of the preceding vehicle **(A)** and bounding box becomes longer.

I assume that the size change of bounding box changed the average distance of matched keypoints.

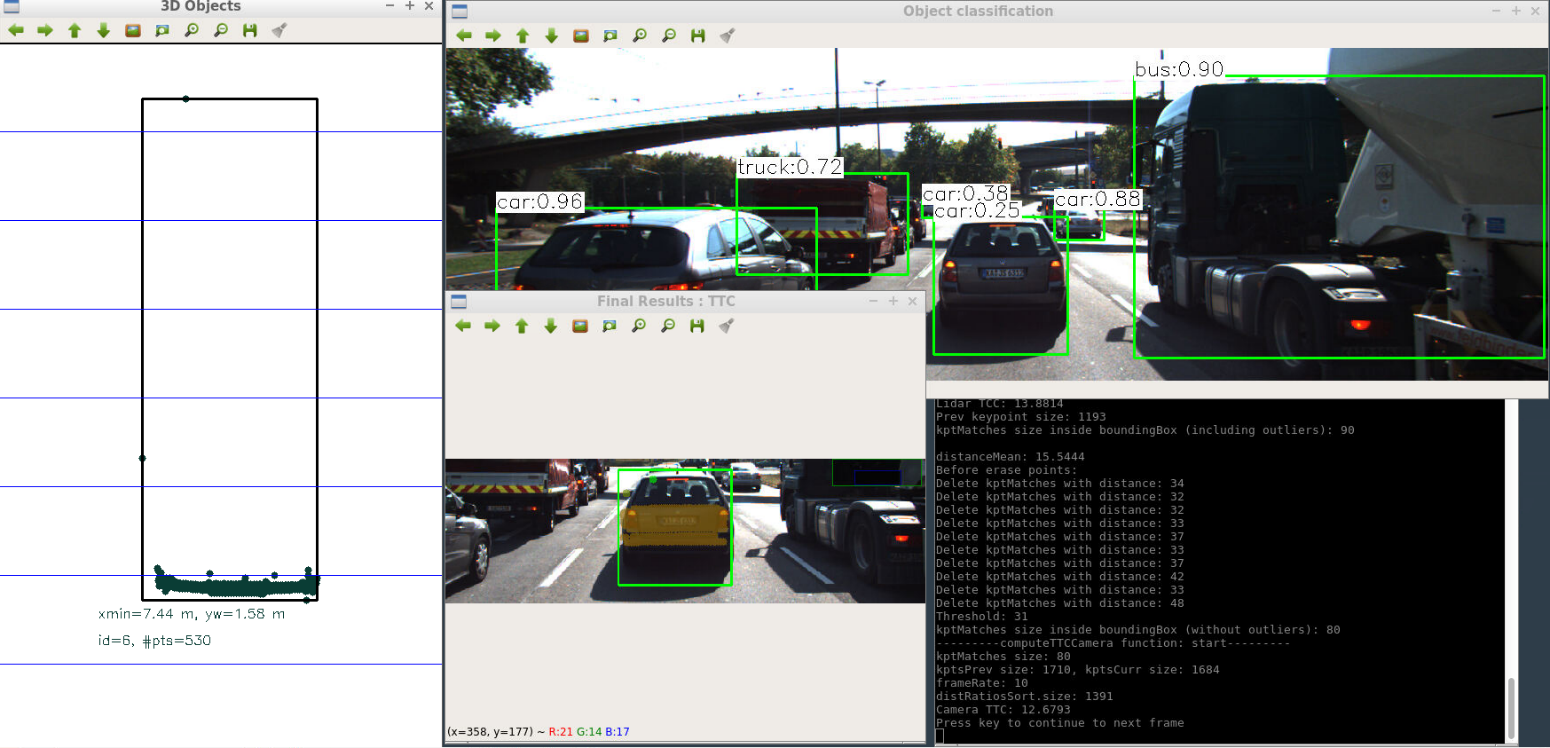
****

**A**

**imageID=5**

**CameraTTC = 28.7596**

Once the size of bounding box keeps same, CameraTTC comes back to normal as shown below.



imageID=6

CameraTTC = 12.6793

**(2)Compare Camera TTC with different kinds of combinations**

The task is complete once all detector / descriptor combinations implemented in previous chapters have been compared with regard to the TTC estimate on a frame-by-frame basis. To facilitate the comparison, a spreadsheet and graph should be used to represent the different TTCs.

From the result of my Project 2 below, I should choose the combinations that can calculate detection & description faster than 100ms (faster than 10 Hz) to execute sensor fusion stably.

I will try the following red-circled 6 combinations.



At first, I show the resulting Lidar TTC below.

It’s clear that Lidar TTC don’t change according to the combinations of detector & descriptor.



Next, Camera TTC is shown below.

The results of detector=ORB are not stable because some of TTCs become too high or too low.

So, there are only 3 choices remaining.



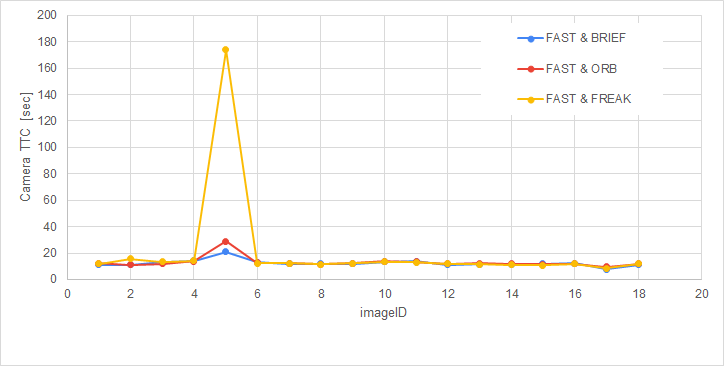
**A**

I saved the Excel file in “result/Project3\_result.xlsx”

Finally, I will choose the best combination from 3 choices.

Detector: FAST & descriptor: FREAK has one unstable TTC, so it’s not the best one. **(A)**

FAST & BRIEF and FAST & ORB are almost same, but FAST & BRIEF is a little better. **(B)**



**B**

**A**

**As a conclusion, “detector: FAST & descriptor: BRIEF” is the best combination in Project3.**