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

**Write up** 1st submit:October 19th, Kenta Kumazaki

**1. Background**

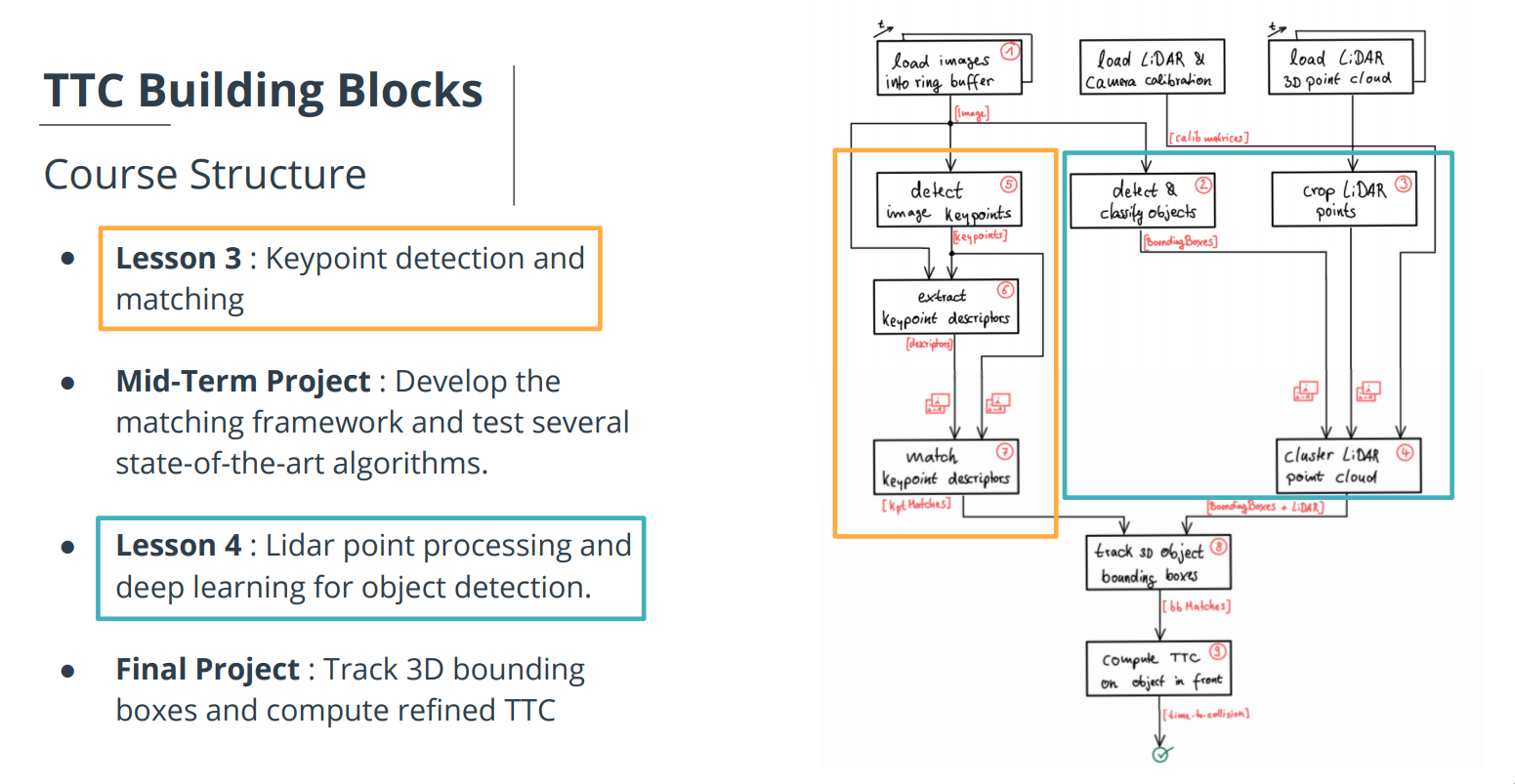
By completing all the lessons, I learned keypoint detectors, descriptors, and methods to match them between successive images. Also, I know how to detect objects in an image using the YOLO deep-learning framework.

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, you will implement the missing parts in the schematic. To do this, you will complete 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**
  + **Project2\_result.xlsx**: The resulting list of calculating TTC.
  + **detector\_\*\*\*\_descriptor\_\*\*\*.txt**: The result of calculation with each combination of detectors/descriptors.

**4. Reflection**

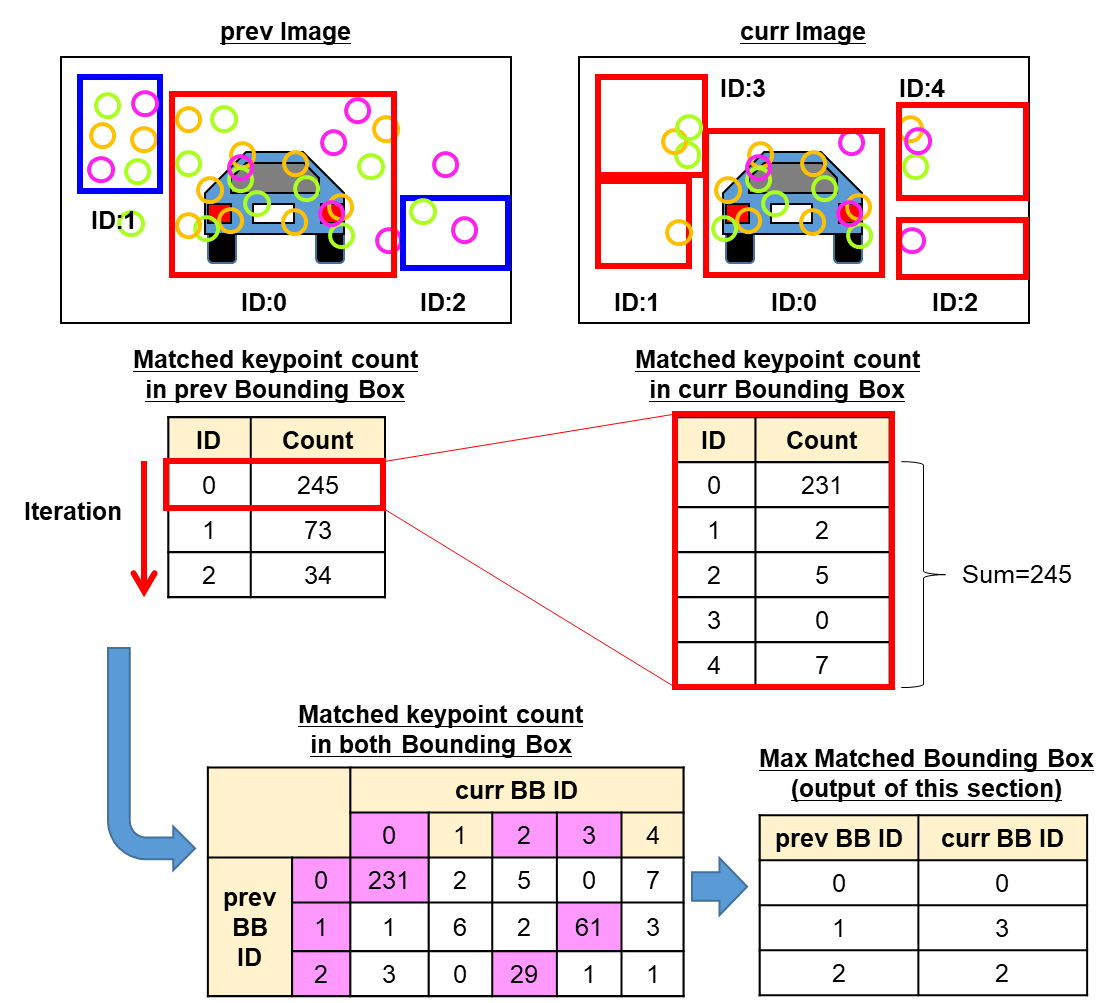
**(1) Match 3D Objects**

**Task FP.1**

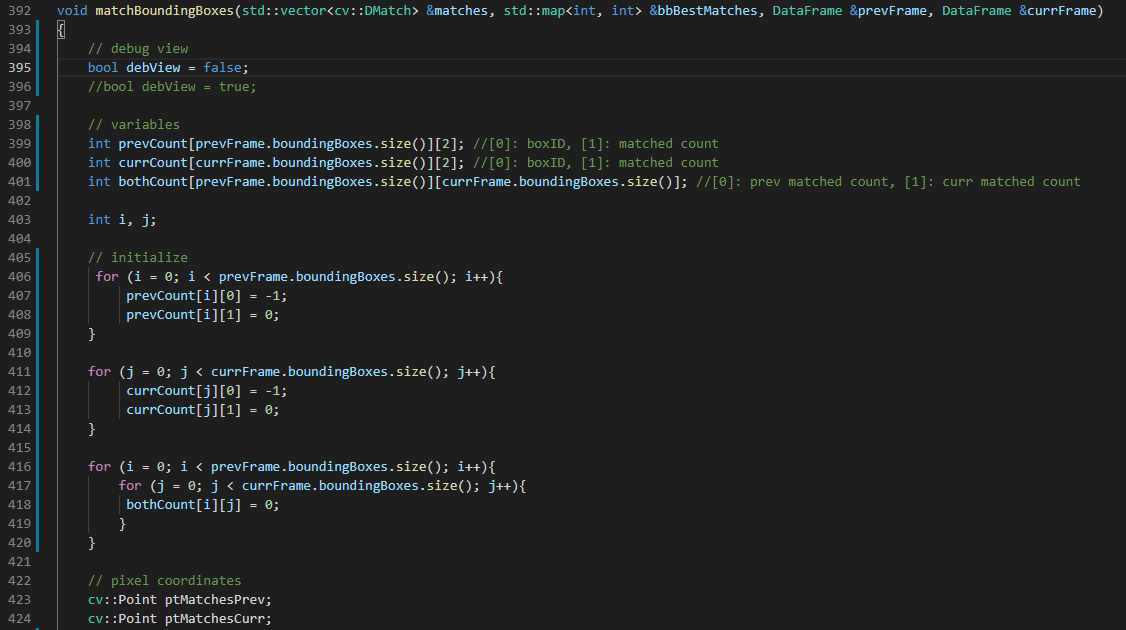
In this task, I implemented the method "matchBoundingBoxes", which takes as input both the previous and the current data frames and provides as output the ids of the matched regions of interest (i.e. the boxID property)“. Matches must be the ones with the highest number of keypoint correspondences.

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:

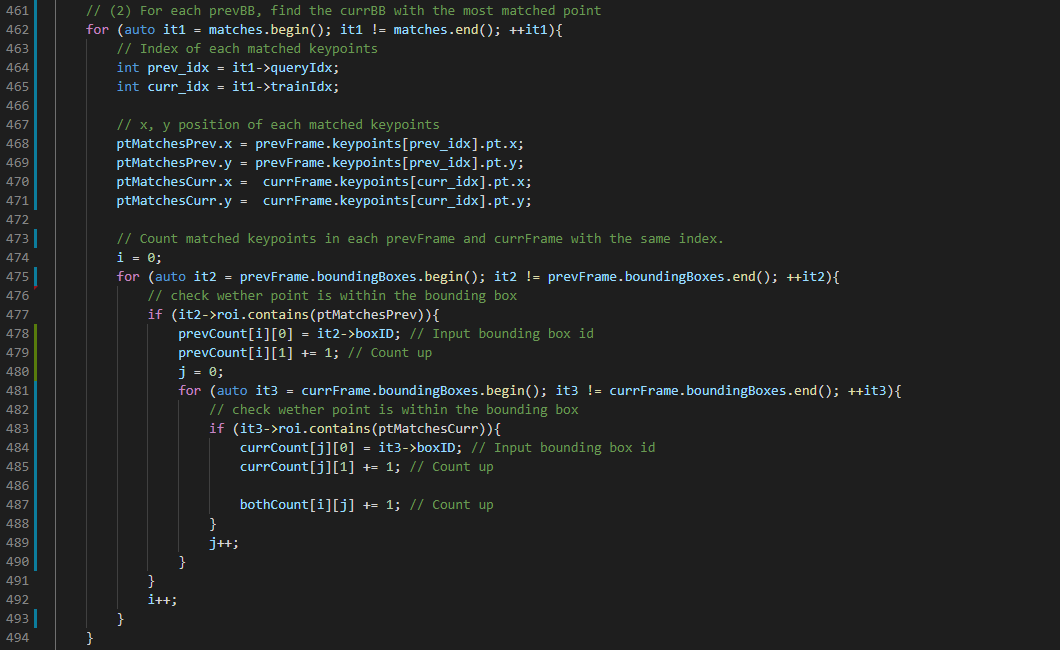


The code of the method "matchBoundingBoxes" is shown below. (camFusion\_Student.cpp)



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)

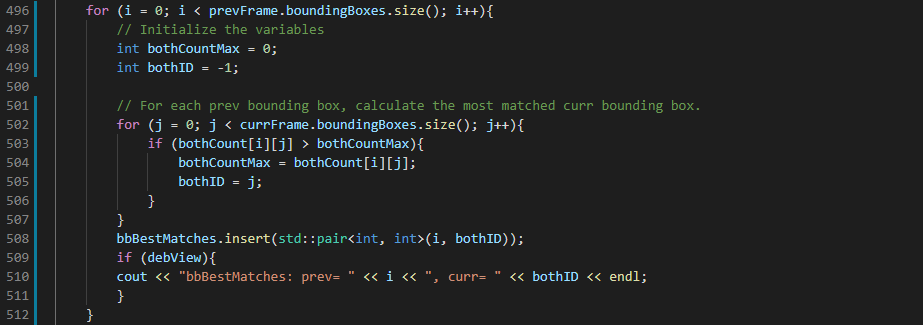


**A**

**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”.

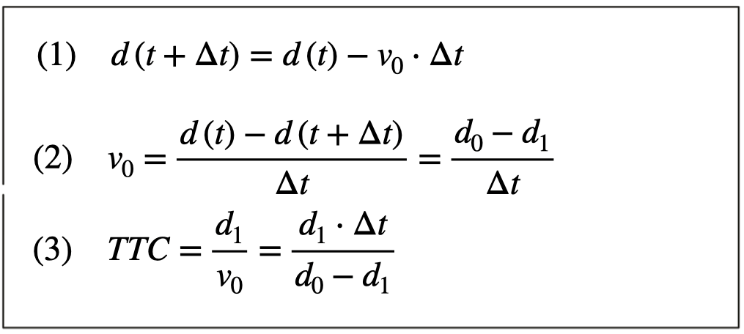


**A**

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

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

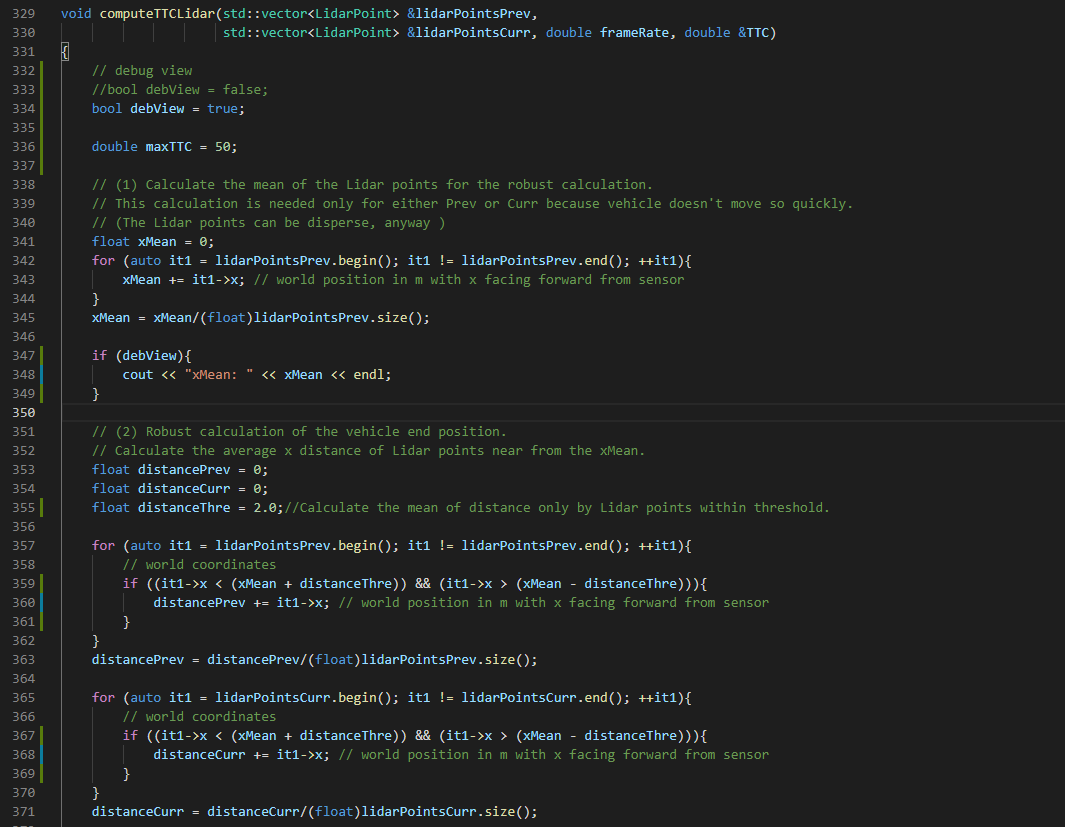
In this part of the final project, my task is to compute the time-to-collision for all matched 3D objects based on Lidar measurements alone. I referred to the "Lesson 3: Engineering a Collision Detection System" of this course to revisit the theory behind TTC estimation show as below.



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.

My code is show below. At first, I set the maxTTC to avoid minus TTC when the current distance is larger than the previous distance. **(A)**

I also set the



**B**

**A**

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

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

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