**Project2: Camera Based 2D Feature Tracking**

**Write up** 1st submit:October 14th, Kenta Kumazaki

**1. Goal**

In this "Feature tracking" project, I implemented a few detectors, descriptors, and matching algorithms.

The aim of implementing various detectors and descriptors combinations is to learn a wide range of options available in the OpenCV library.

**2. Overview**

The steps of this project are the following:

1. **The Data Buffer**: I focused on loading images, setting up data structures and putting everything into a ring buffer to optimize memory load.
2. **Keypoint Detection**: I integrated several keypoint detectors such as HARRIS, FAST, BRISK and SIFT and compare them with regard to number of keypoints and speed.
3. **Descriptor Extraction & Matching**: I focused on descriptor extraction and matching using brute force and also the FLANN approach.
4. **Performance Evaluation**: Once the code framework is complete, I tested the various algorithms in different combinations and compare them with regard to some performance measures.

**3. Submission**

**(1) GitHub**

<https://github.com/kkumazaki/Sensor-Fusion_Project2_Camera-Based-2D-Feature-Tracking.git>

**(2) Directory**

I cloned the basic repository from Udacity <https://github.com/udacity/SFND_2D_Feature_Tracking>

and added/modified the following files.

* **Writeup\_of\_project2.pdf**: This file
* **README.md**: Read me file of this repository
* **src**
  + **MidTermProject\_Camera\_Student.cpp**: Main script to set the initial conditions and run the functions.
  + **matching2D\_Student.cpp**: Script used to create the functions of detectors, descriptors and matchers.
* **result**
  + **Project2\_result.xlsx**: The resulting list of calculating keypoint numbers and time to execute   
     of all possible combinations with detectors/descriptors.
  + **detector\_\*\*\*\_descriptor\_\*\*\*.txt**: The result of calculation with each combination of detectors/descriptors.
  + **detector\_\*\*\*\_descriptor\_\*\*\*.jpg**: The image of matching the 1st and 2nd frame with each combination   
     of detectors/descriptors.

**4. Reflection**

**(1)The Data Buffer**

**Task MP.1**

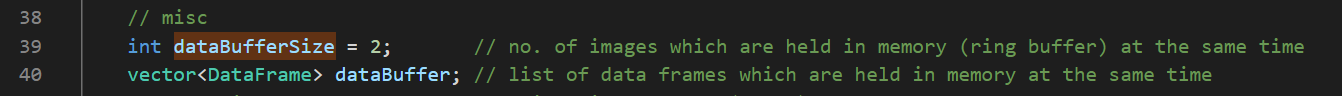
I focused on loading images, setting up data structures and putting everything into a ring buffer to optimize memory load. I modified “MidTermProject\_Camera\_Student.cpp” as following.

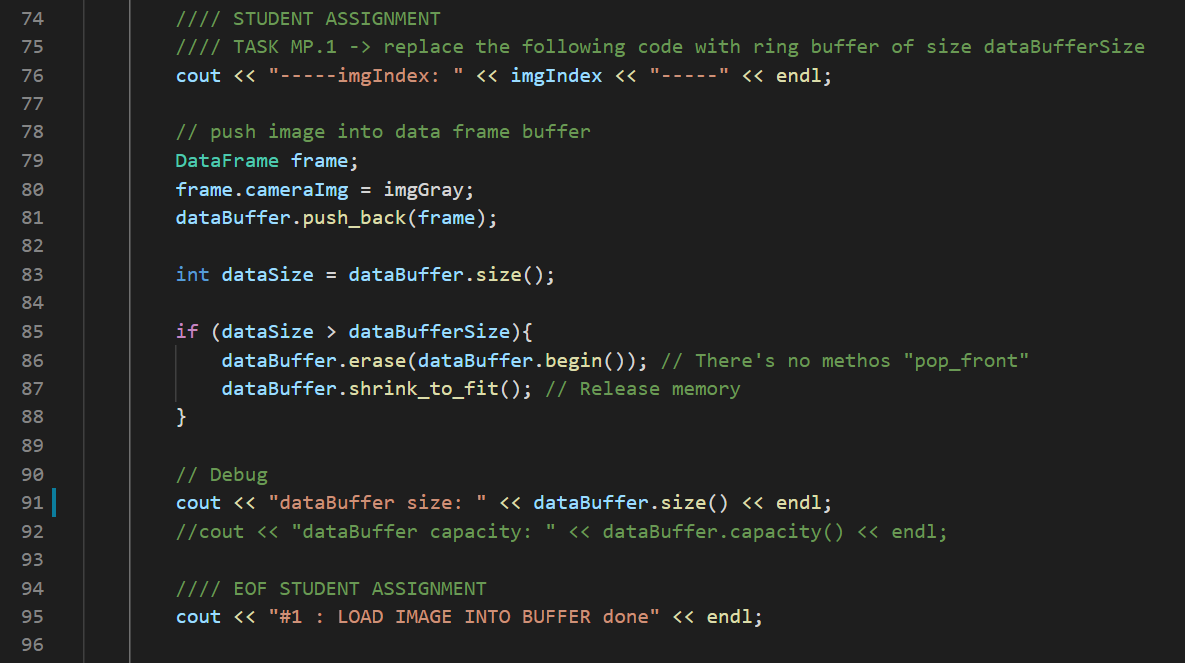
I added the logic to erase the oldest dataBuffer if data size of buffer is bigger than specified size. **(A)**

In this case, the specified size is 2 because we only need 2 pairs to execute matching process. **(B)**

As a result, I checked the buffer size never becomes more than 2. **(C)**

**B**





**C**

**A**

**(2)Keypoint Detection**

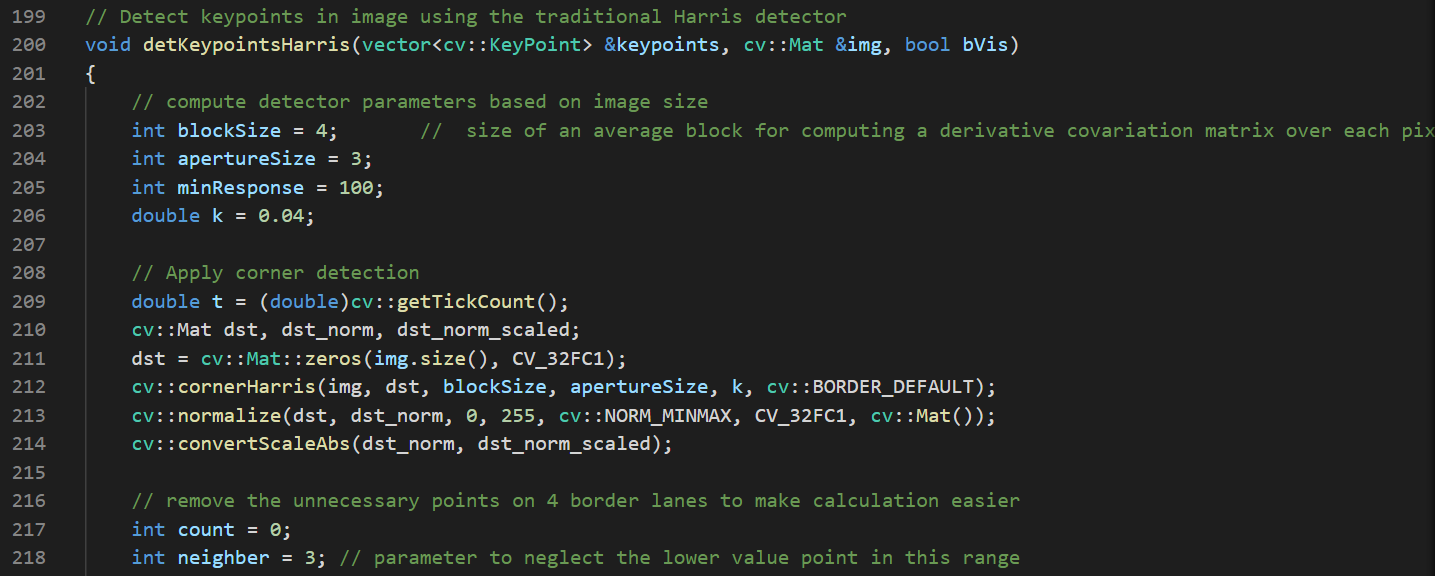
**Task MP.2**

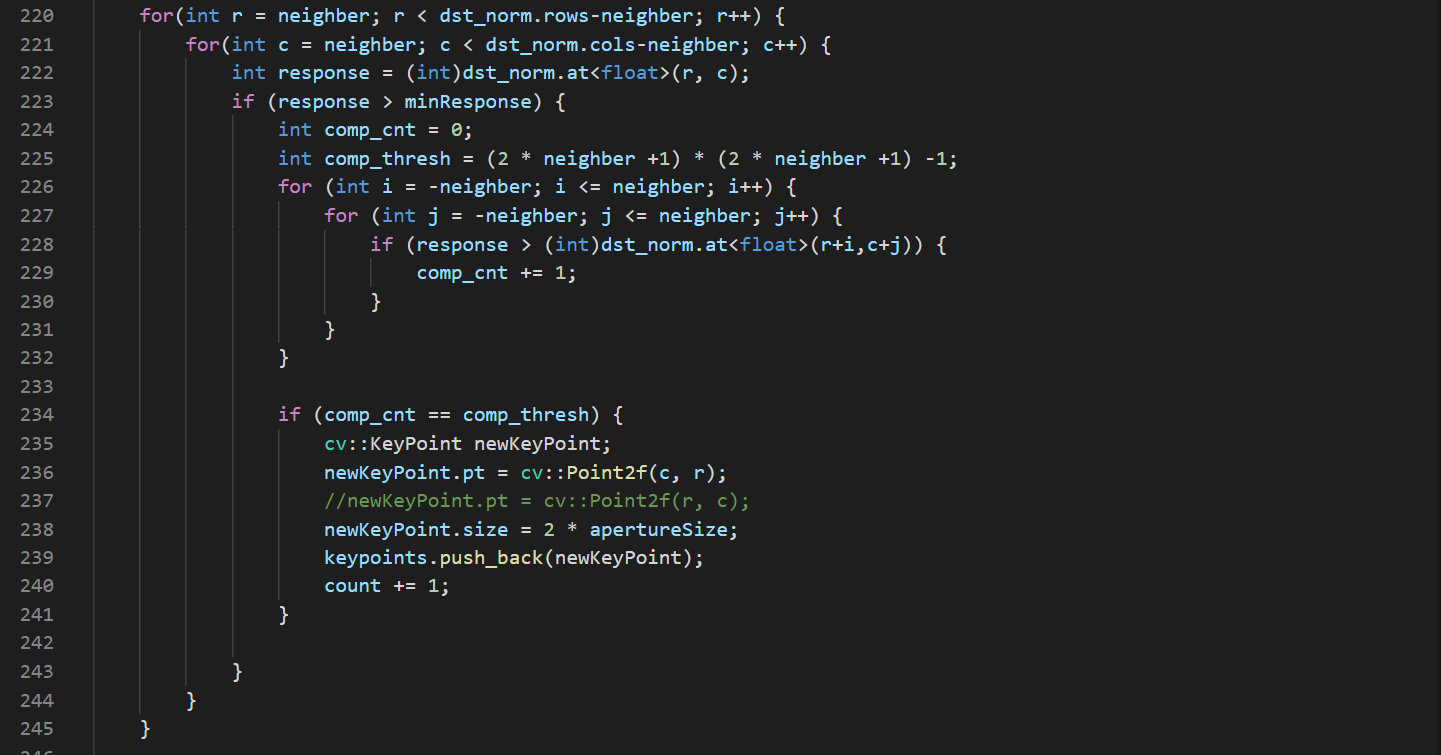
My second task is to focus on keypoint detection.

In the student version of the code there’s already an existing implementation of the Shi-Tomasi detector.

I implemented a selection of alternative detectors, which are HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT  
in “matching2D\_Student.cpp” as following.

HARRIS detector is a traditional method, so there’s the function only for HARRIS as below.





Other detectors are modern methods, so they are implemented in one function as below.

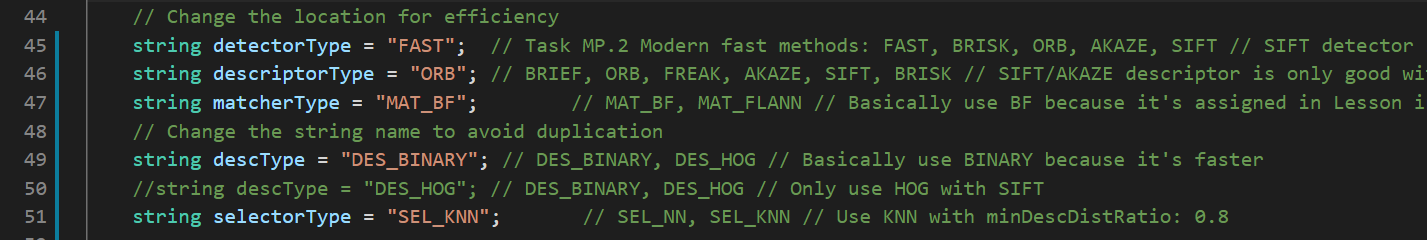
I use IF Statements to switch the detector type.



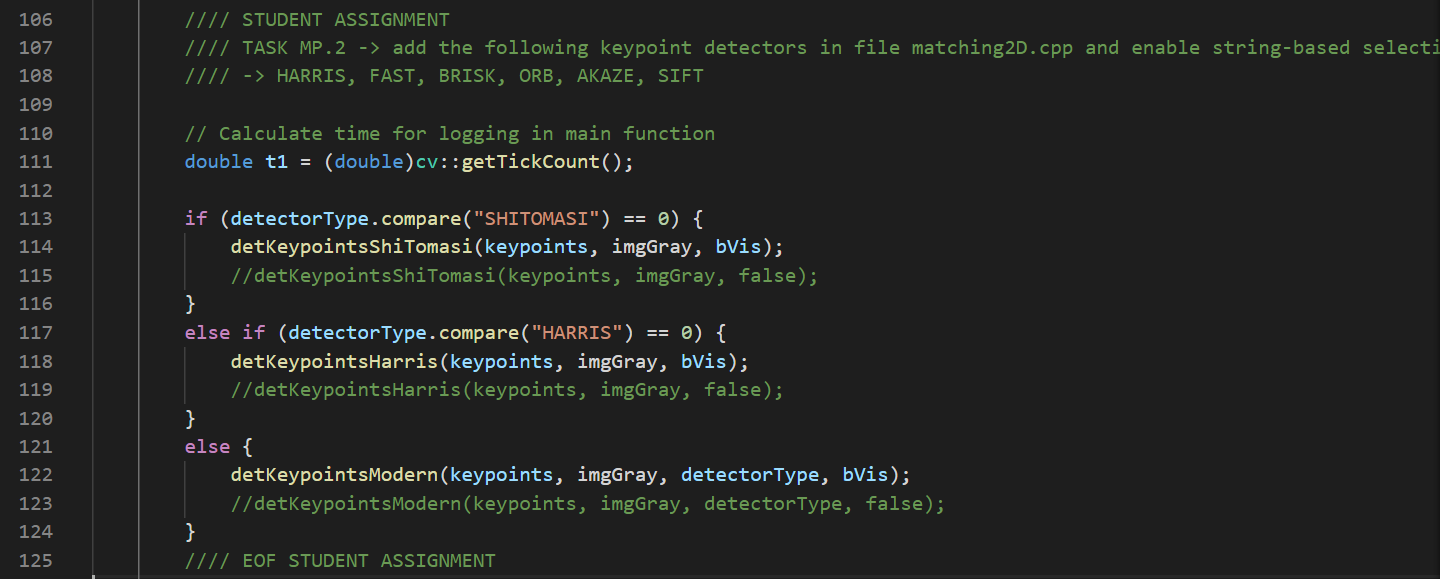
There are some codes to select the type of detector in “MidTermProject\_Camera\_Student.cpp” as below.

For logging purpose in Task MP.7, 8, 9, I changed the location of the selection in the beginning of Main Function.

(Same thing to the selection of descriptors and matchers)



In the Main Function, I use the 3 functions to calculate detection of keypoints as following.



**Task MP.3**

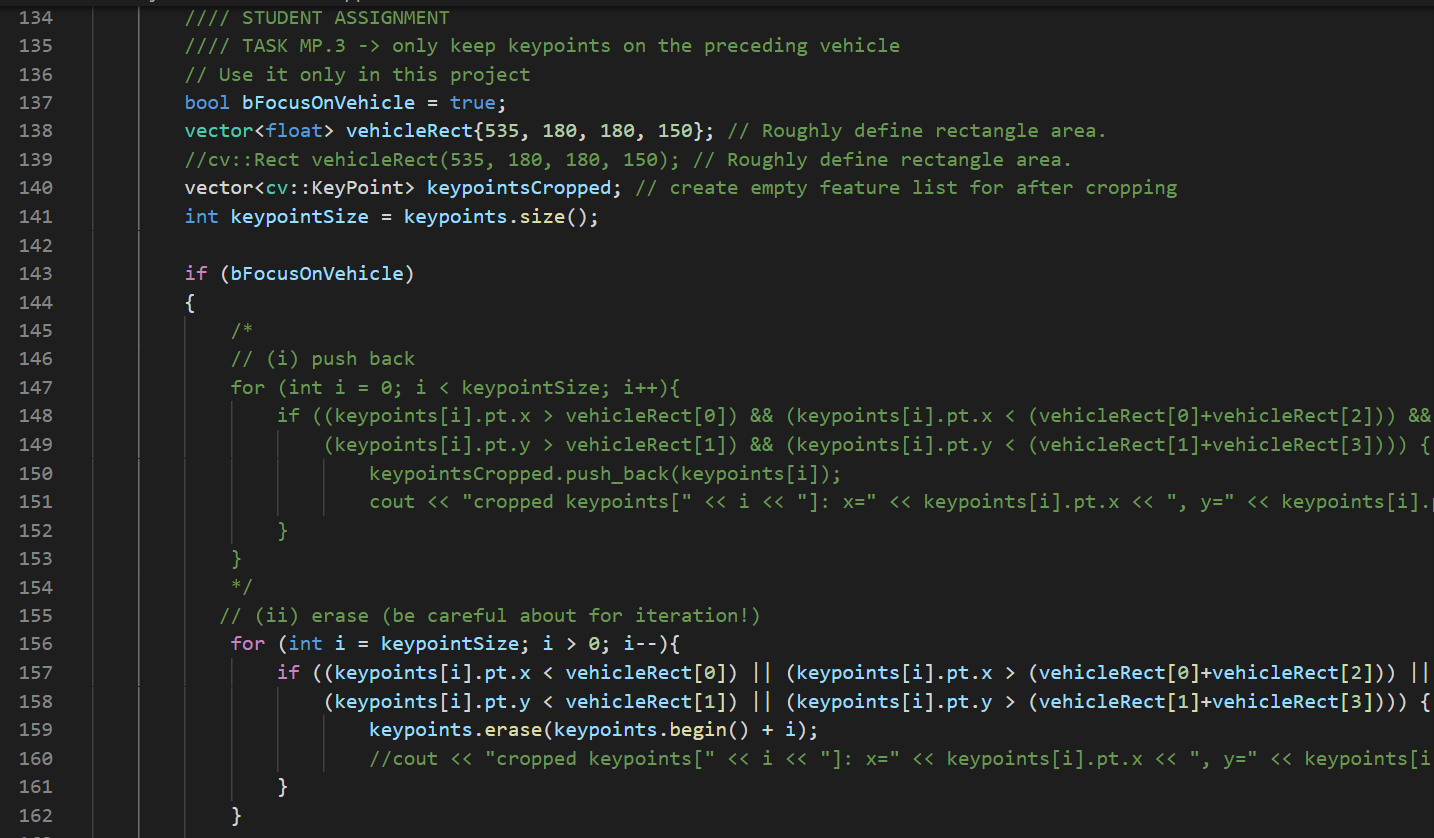
My third task is to remove all keypoints outside of a bounding box around the preceding vehicle.

Box parameters you should use are : cx = 535, cy = 180, w = 180, h = 150.

I implemented it in “MidTermProject\_Camera\_Student.cpp” as following.

At first I tried to create another KeyPoint object and push\_bach the cropped keypoints, but I found that it’s wasting the memory resources. **(A)**

Instead of that, I erased the keypoints, whose positions are out of range, from the original KeyPoint object. **(B)**



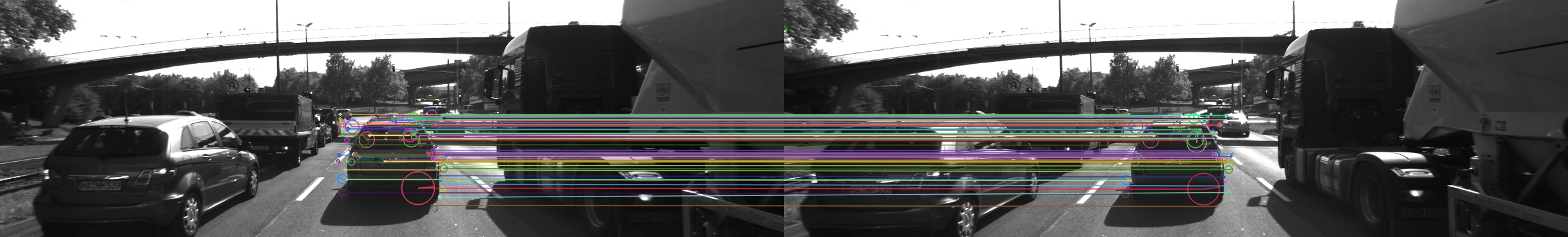
**B**

**A**

The resulting keypoints are shown below. (e.g. detector = SIFT)

I was able to remove the keypoints outside the specified range. **(C)**

**x**



**150**

**180**

**(535, 180)**

**y**

**C**

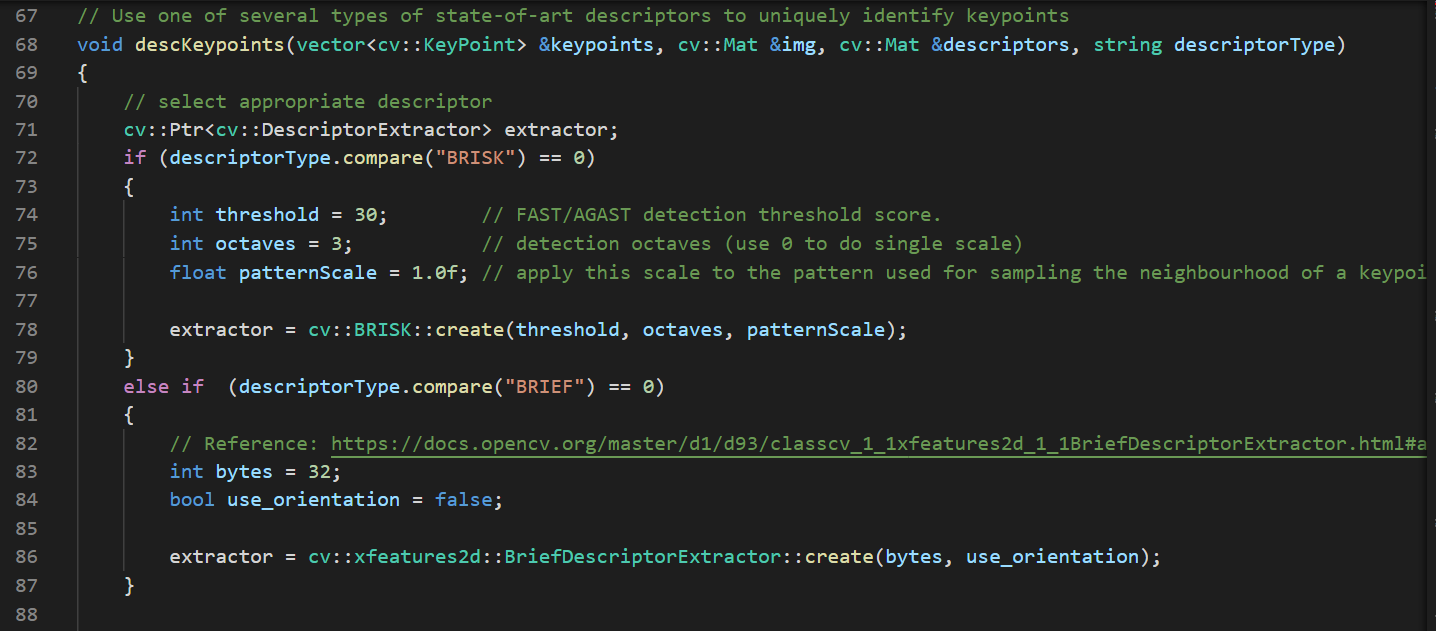
**(3)Descriptor Extraction & Matching**

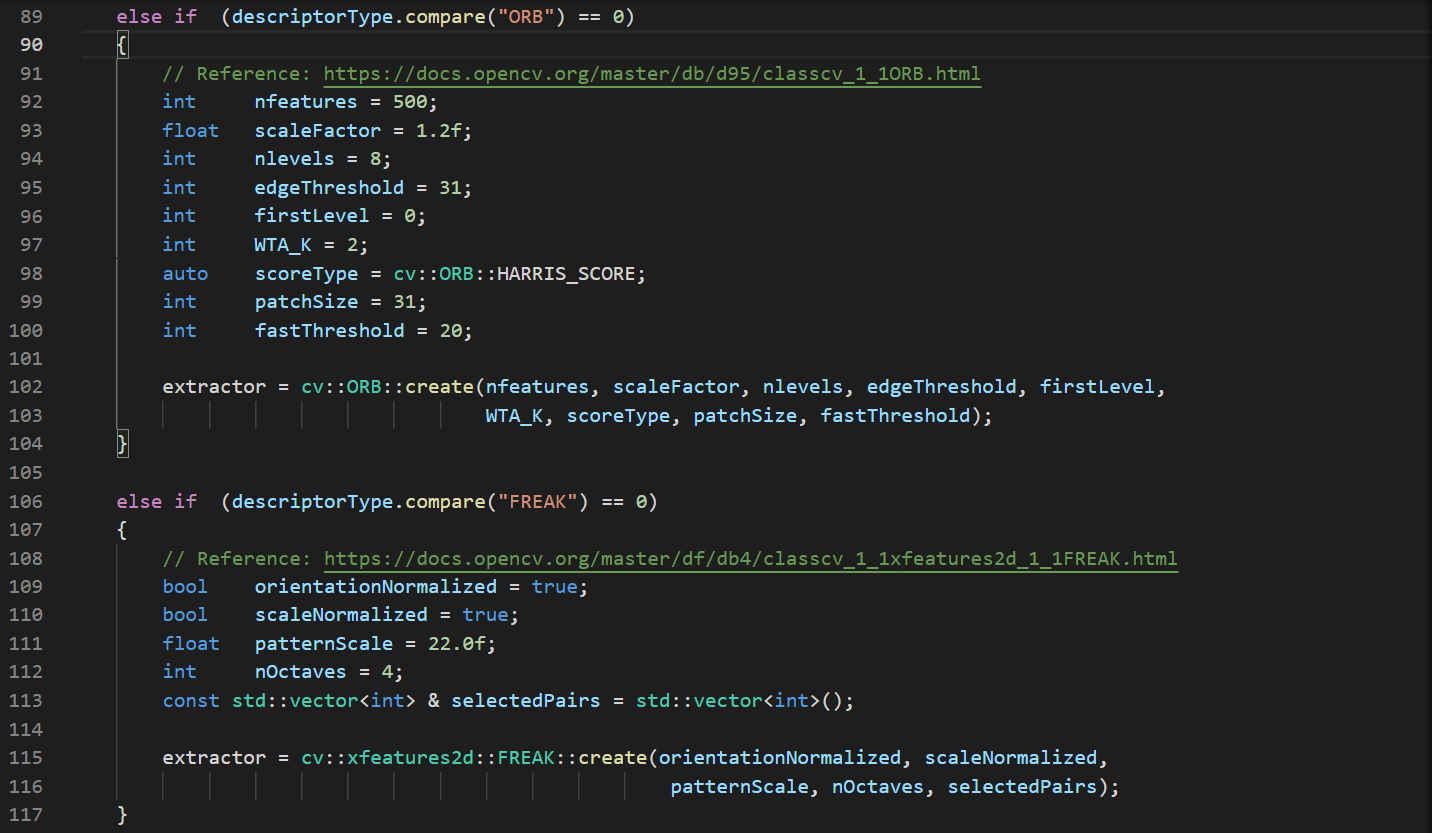
**Task MP.4**

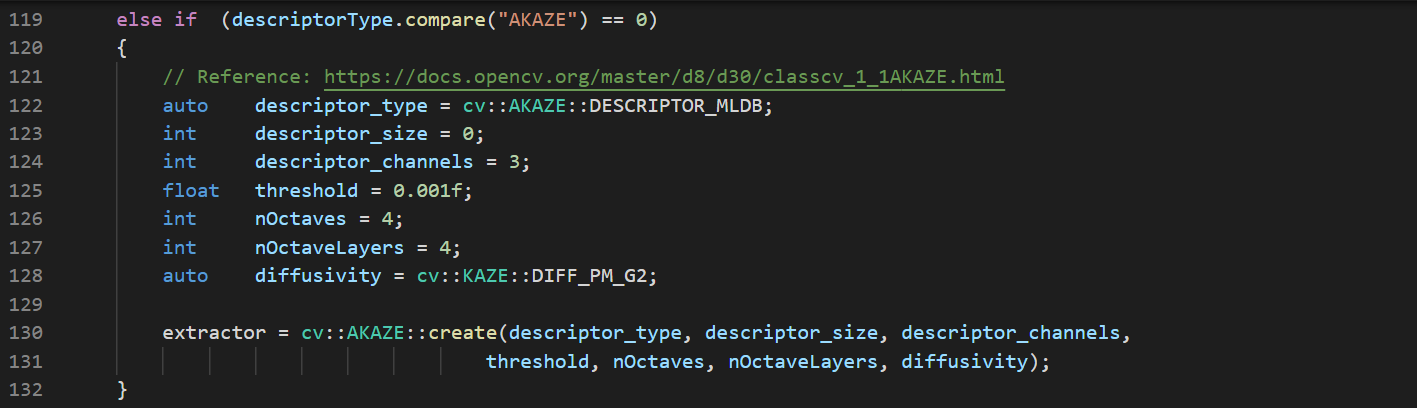
My fourth task is to implement a variety of keypoint descriptors to the already implemented BRISK method and make them selectable using the string 'descriptorType'. The methods I must integrate are BRIEF, ORB, FREAK, AKAZE and SIFT. The SURF is not a part of the mid-term project.

I implemented the source codes of each descriptor in “matching2D\_Student.cpp” as below.

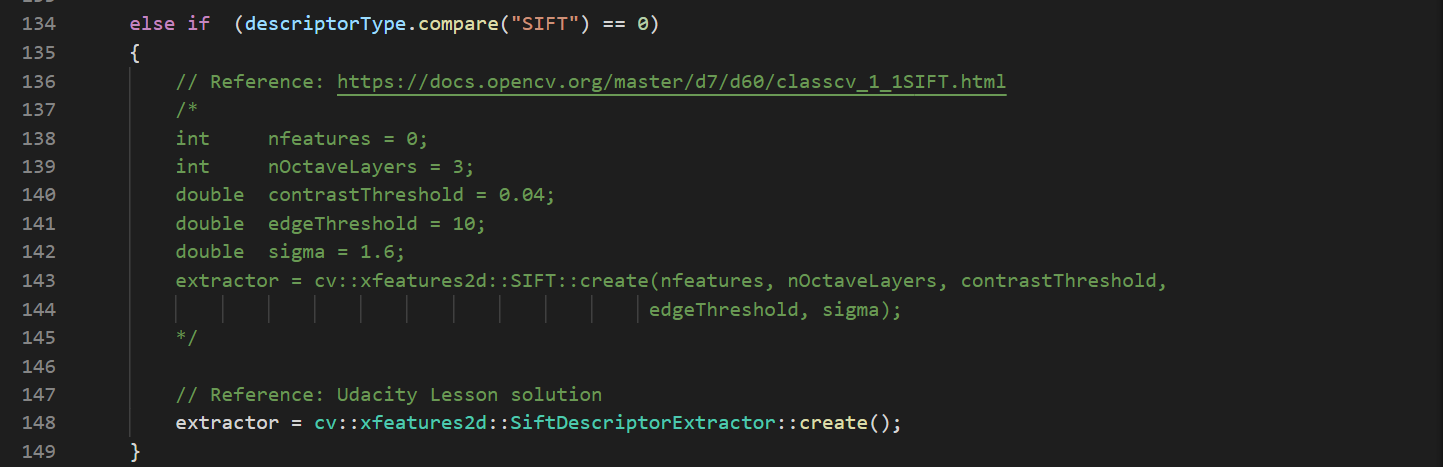
I mainly referred to OpenCV reference pages.





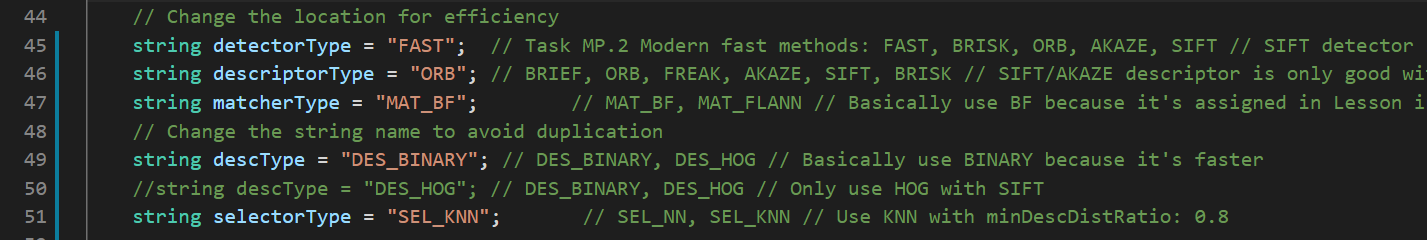


I referred to the solution source codes in Udacity Lesson for SIFT as below.

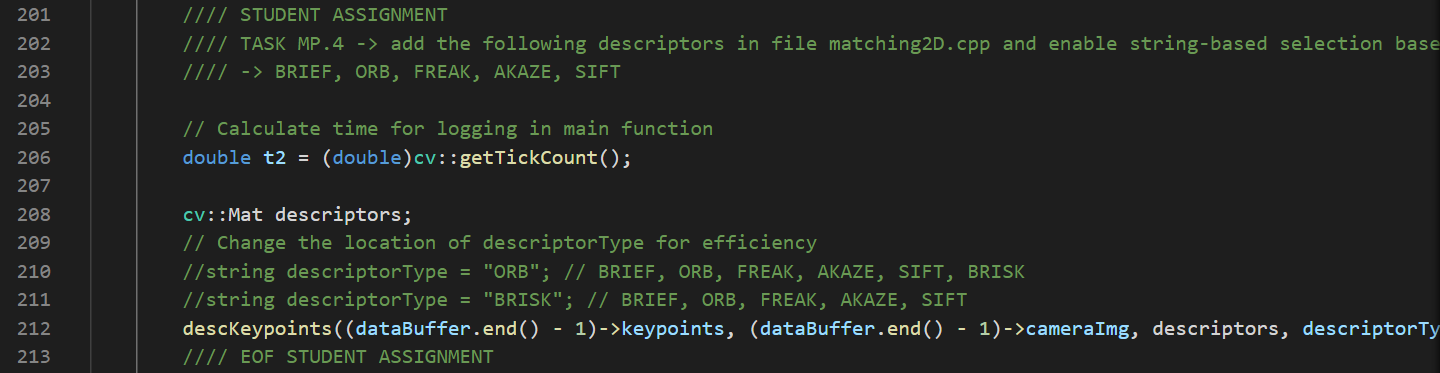


There are some codes to select the type of descriptor in “MidTermProject\_Camera\_Student.cpp” as below.

As already written in detector section, there’s the selection part in the beginning of Main Function.



Same as detector, in main function I run the function to calculate description as following.

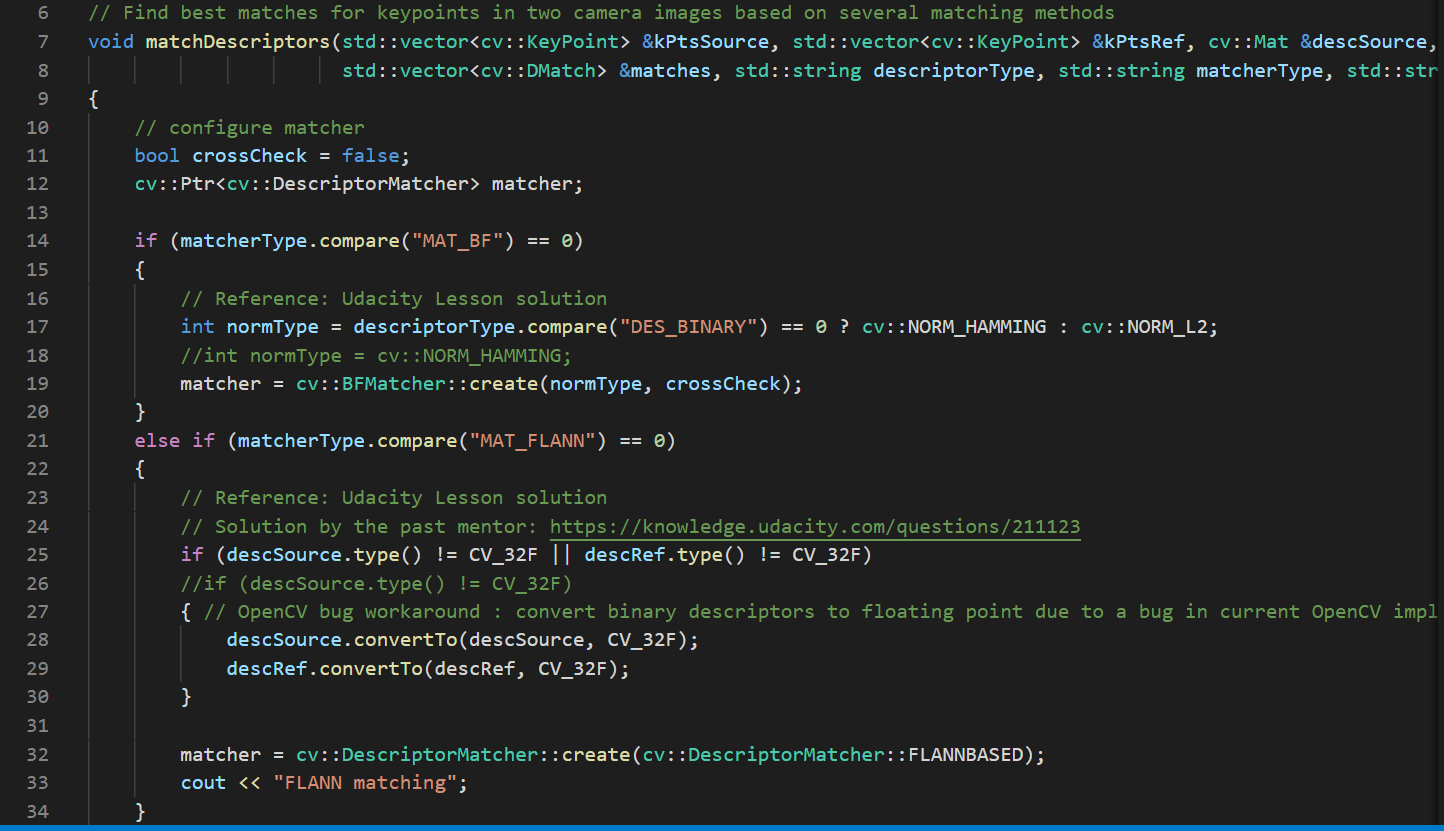


**Task MP.5**

My fifth task focuses on the matching part. The current implementation uses Brute Force matching   
combined with Nearest-Neighbor selection. I must now add FLANN as an alternative to brute-force as well as the K-Nearest-Neighbor approach.

I implemented the source codes of each matcher and selector in “matching2D\_Student.cpp” as below.

To implement FLANN, I mainly referred to the solution source codes in Udacity Lesson. **(A)**

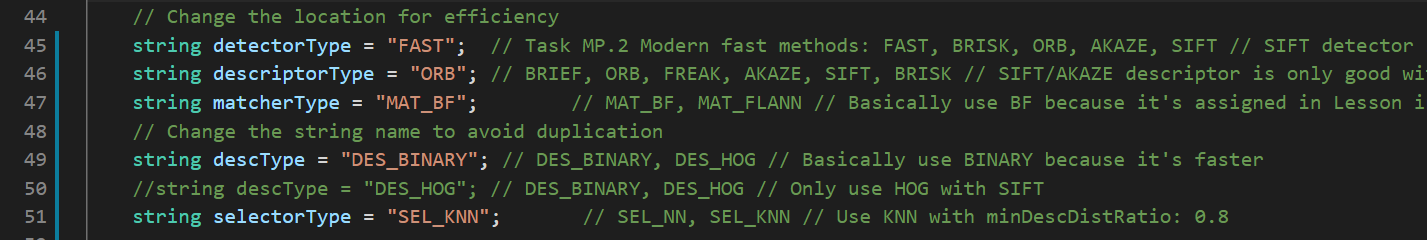


**A**

There are some codes to select the type of matcher in “MidTermProject\_Camera\_Student.cpp” as below.

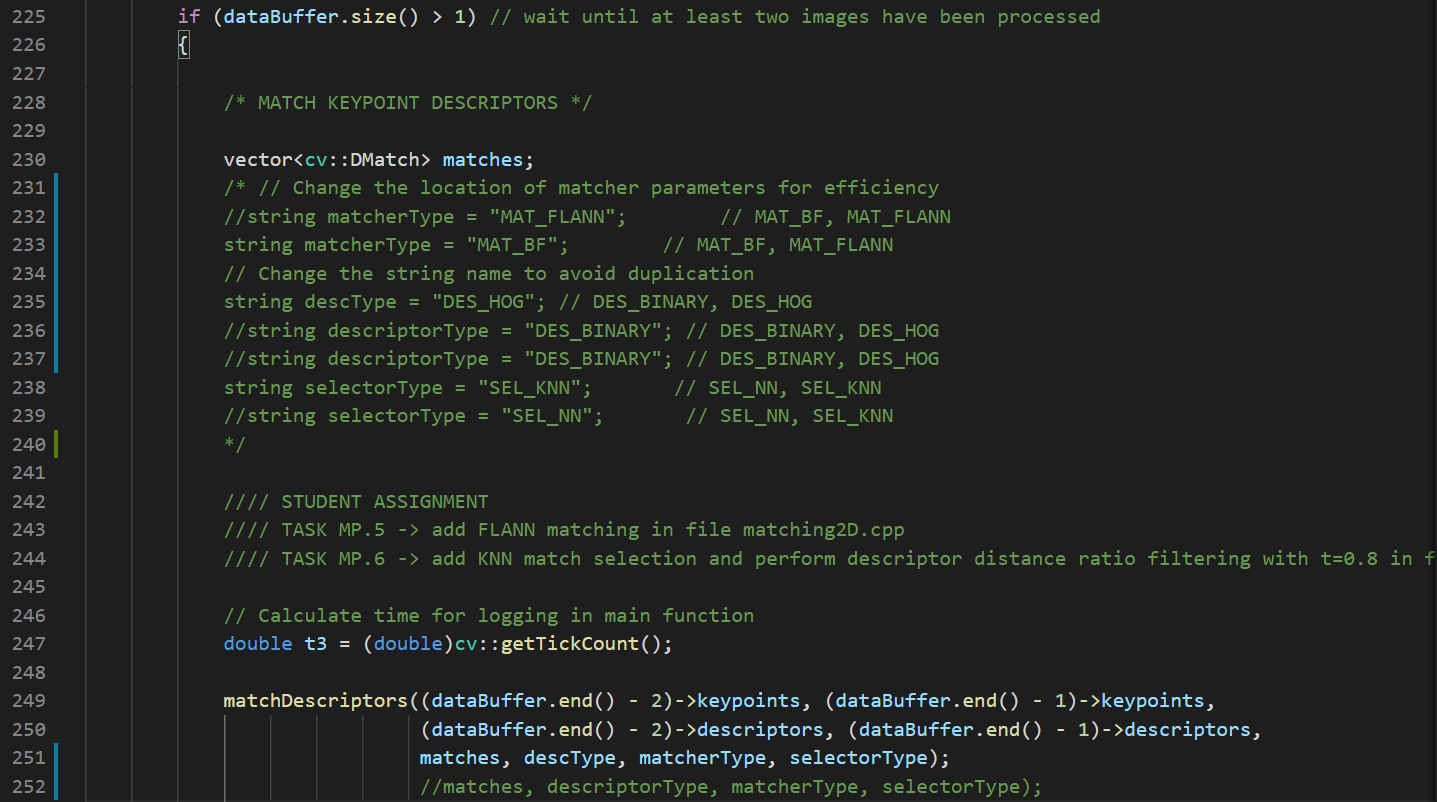
As already written before, there’s the selection part in the beginning of Main Function.

The “descriptorType” becomes duplication, so I changed the variable name as following. (B)



**B**

Matcher needs more than 1 image frame, so it runs only when dataBuffer size is more than 1.

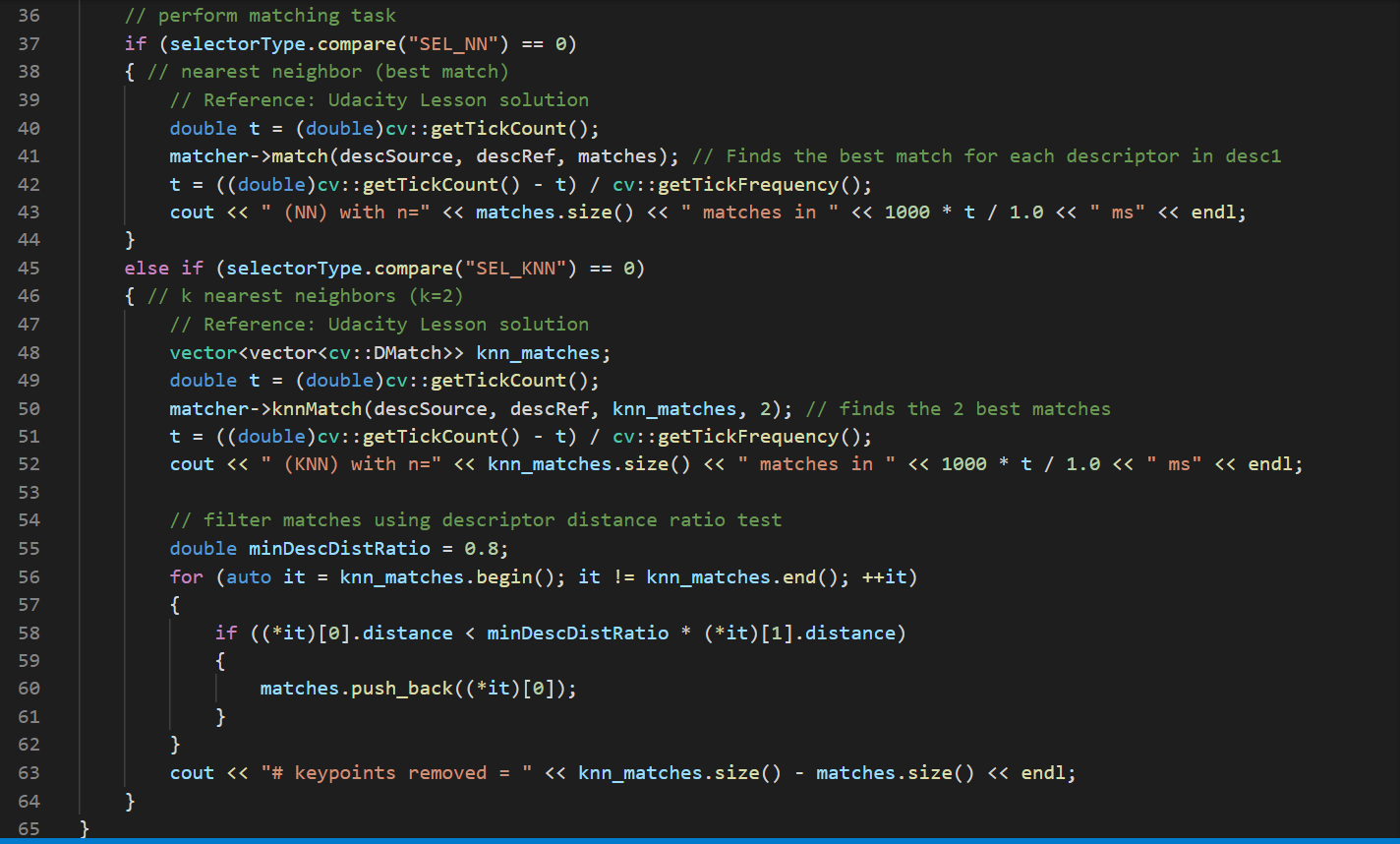


**Task MP.6**

As my sixth task, I then implement the descriptor distance ratio test as a filtering method to remove  
bad keypoint matches.

I implemented the source codes of each matcher and selector in “matching2D\_Student.cpp” as below.

To implement KNN and filter matches using descriptor distance ratio test, I mainly referred to the solution source codes in Udacity Lesson. **(A), (B)**



**A**

**B**

**(4)Performance Evaluation**

**Task MP.7**

My seventh task is to count the number of keypoints on the preceding vehicle for all 10 images and take note of the distribution of their neighborhood size. Do this for all the detectors you have implemented.

I followed the mentor's instruction "SIFT detector only works well with SIFT descriptor whose type is DES\_HOG".

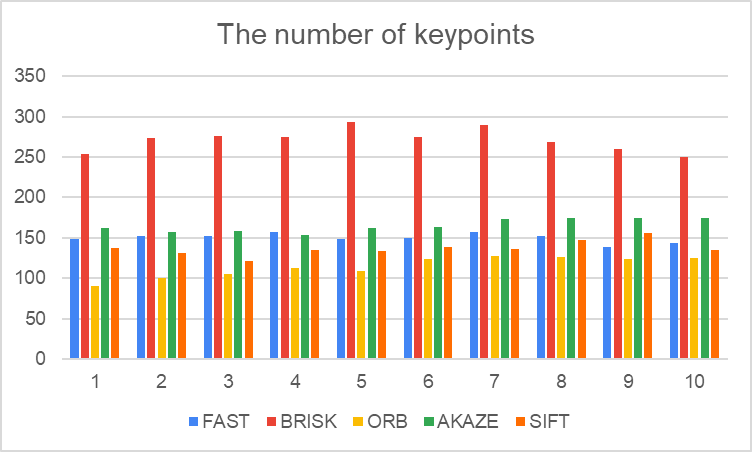
I used DES\_HOG only for this combination because DES\_BINARY is faster.

<https://knowledge.udacity.com/questions/323235>

The number of keypoints with each detector is shown below.

BRISK has the most number, and ORB has the smallest.





I take note of the distribution of their neighborhood size of each detector as below.

As written in the mentor’s instruction, I just check them in each image file. (No Calculation)

<https://knowledge.udacity.com/questions/106021>

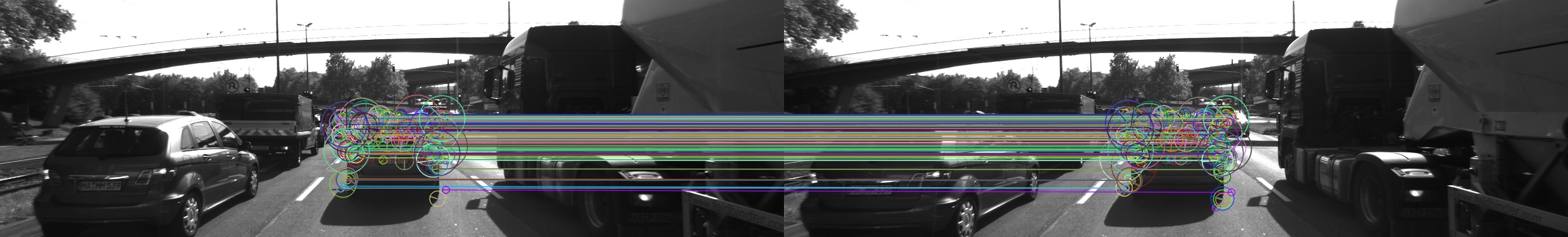
FAST and AKAZE have mostly small neighborhood sizes, and the dispersion looks small.

BRISK and SIFT have small and large neighborhood sizes, and the dispersion looks large.

ORB have mostly large neighborhood sizes. Only ORB has the mismatched keypoint on the tree. **(A)**

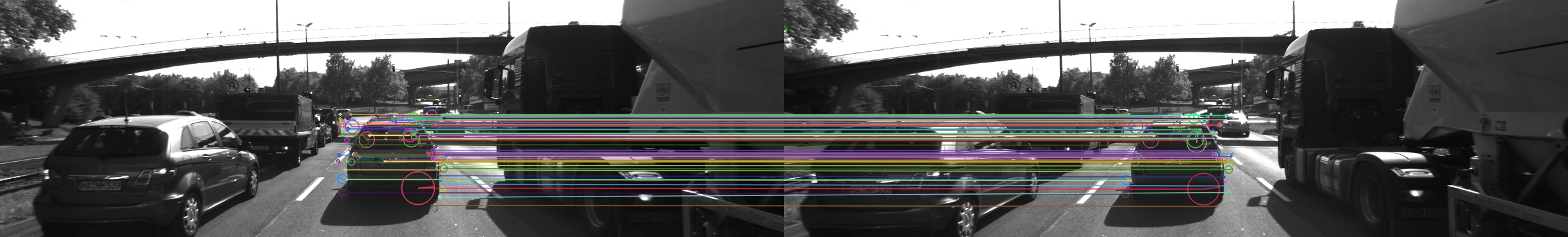
Overall, **FAST** and **AKAZE** look good detector compared with others.

<FAST> <BRISK> <ORB>

**A**

<AKAZE> <SIFT>

**Task MP.8**

My eighth task is to count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors.

In the matching step, use the BF approach with the descriptor distance ratio set to 0.8.

As written above, I ran SIFT detector only with SIFT descriptor to avoid error.

I also followed the mentor's instruction "AKAZE descriptor only works well with AKAZE detector.

(We need to implement AKAZE detector with other descriptors, though)

<https://knowledge.udacity.com/questions/163998>

I show the result of number of matched keypoints as following.



**Task MP.9**

My ninth task is to log the time it takes for keypoint detection and descriptor extraction.

The results must be entered into a spreadsheet and based on this information I then suggest the TOP3 detector / descriptor combinations as the best choice for our purpose of detecting keypoints on vehicles.

Finally, in a short text, I justify my recommendation based on my observations and on the data I collected.

I used the same matcher (BF) and selector (KNN with minDescDistRatio = 0.8) as MP.7, 8.

The summation of detection time and description time is shown below.



**B**

**A**

The time of ORB detector is short **(A)**, but keypoints of ORB are not stable as I wrote in Task MP.7.

Besides ORB, the TOP 3 fast methods are following. **(B)**

1. FAST detector & ORB descriptor: average 2.6 msec, standard deviation: 0.6 msec
2. FAST detector & BRIEF descriptor: average 3.3 msec, standard deviation: 3.7 msec
3. FAST detector & FREAK descriptor: average 45.7 msec, standard deviation: 2.0 msec

(3) is more than 10 times slower than (1), (2), so it’s not a good solution.

(2) is a little slower than (1), but the standard deviation is much larger than (1) so it’s not good either.

**As a conclusion, (1)FAST detector & ORB descriptor is the best combination in this project images.**

The matching image of this combination is shown below. It looks there’s no mismatch.

“result/detector\_FAST\_descripter\_ORB\_image.jpg”

