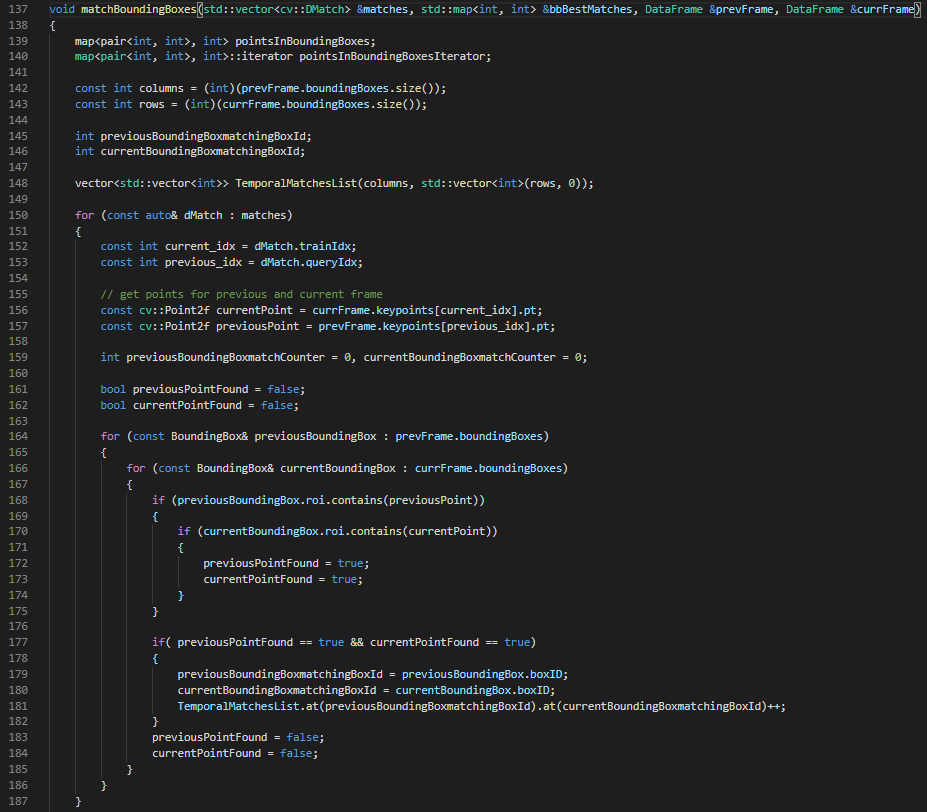
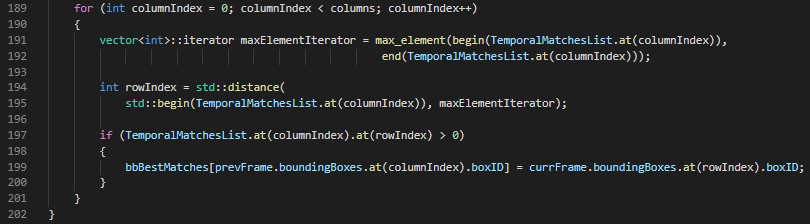
**3D Object Tracking Report**

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..

In the the matchBoundingBoxes function I iterates through all points found in the matches vector, from line 165 to 186 with a for, the implemented program determines if the previous and the current point in question exists, if the previous and current point exist, the count of the occurrence is updated within a temporary vector named “TemporalMatchesList”, If not, the next previous and current point in the matches vector is searched.

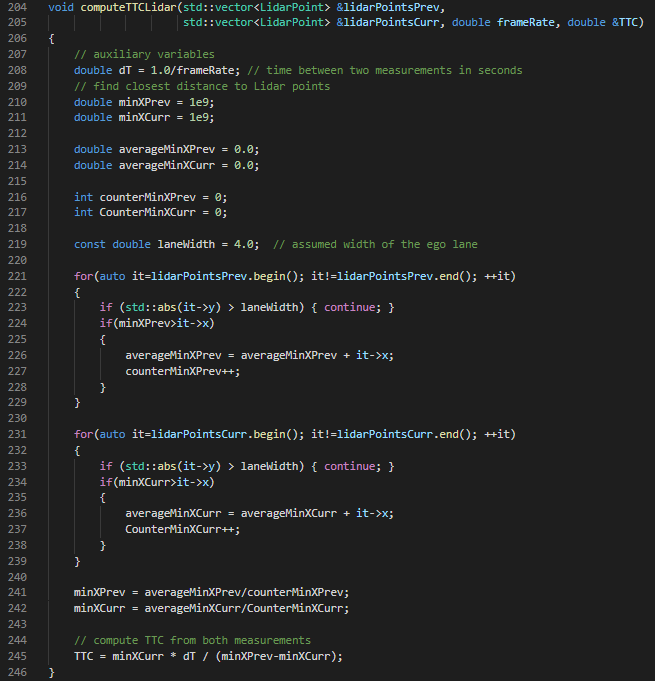


Another for loop from line 189 to 201 is implemented to iterate through all points in the “TemporalMatchesList” vector, and store the best points in the bbBestMatches map.



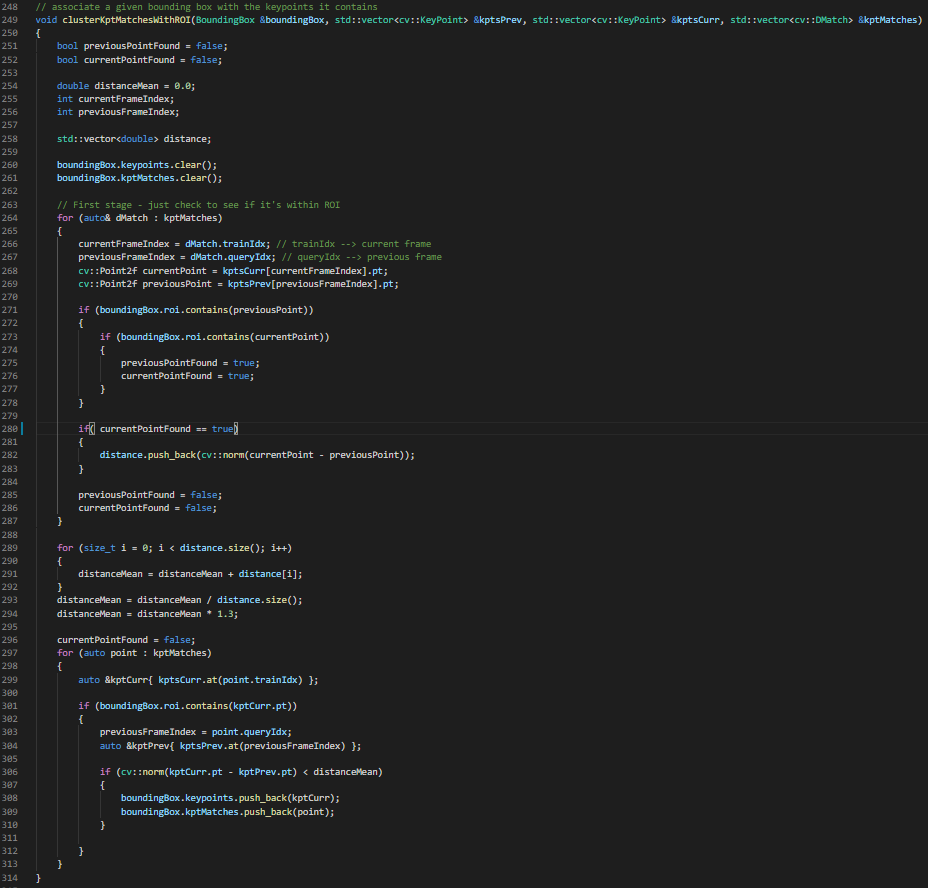
FP.2 Compute Lidar-based TTC: Code is functional and returns the specified output. Also, the code is able to deal with outlier Lidar points in a statistically robust way to avoid severe estimation errors.

From line 221 to 229 the program get the minimum previous point and with “minXPrev” at line 241 calculates the average previous minimum point in order to minimize the noise or errors, from line 231 to 239 a for loop is implemented to do the same but for the minimum current point, at line 242 the “minXCurr” variable calculates the current minimum point in order to minimize the noise. At line 245 the “time to collision (TTC)” is calculated using the minimum previous and corrent average point to avoid severe estimation errors, in the next image you can see the code implemented:



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.

From 249 to 314 line the keypoint maches are clustered and associated with bounding boxes in the images, from 264 to 287 line a foor loop checks if the current points in kptMatches vector exist in the bounding box, if exist it, a distance between the current and previous point are calculated and at 293 line an average distance using all points founded is calculated. At the end from 297 to 313 line a foor loop is implemented and if the current point is founded a distance between the current and previous point is less than the average distance the point is stored, in the next image you can see the code implemented.



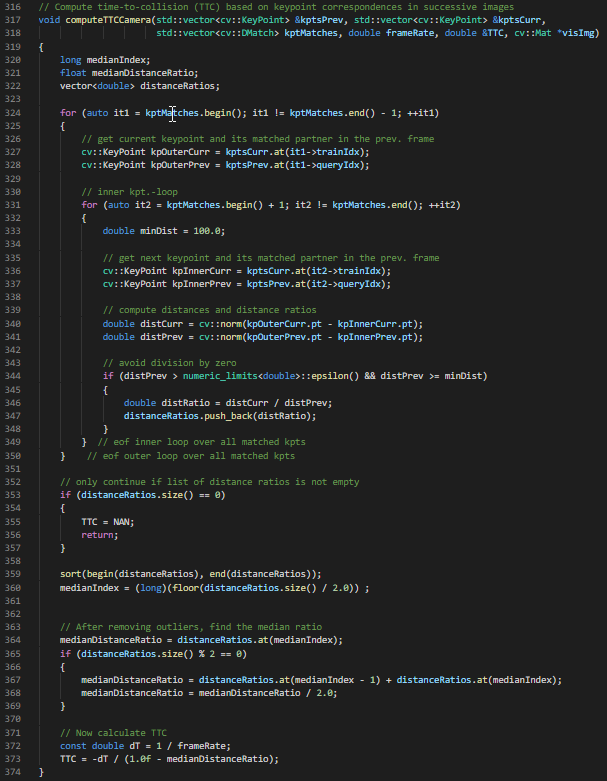
FP.4 Compute Camera-based TTC: Compute the time-to-collision in second for all matched 3D objects using only keypoint correspondences from the matched bounding boxes between current and previous frame.

The code implemented is very similar to the udacity lesson, a foor loop iterates all points in kptMatches vector, the distance between all keypoints on the vehicle is used to estimate the height ratio.

The formula used to calculate the TTC is as follows:

TTC = -dT / (1 - medDistRatio);

The median distance was used to remove any outlier inﬂuence, from 359 to 369 line a median is calculated, you can see the code implemented in the next image:



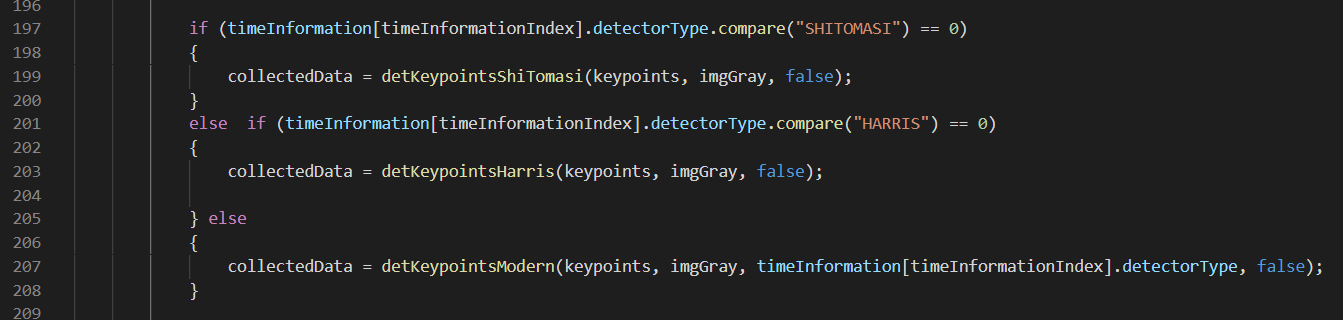
FP.5 Performance Evaluation 1: Find examples where the TTC estimate of the Lidar sensor does not seem plausible. Describe your observations and provide a sound argumentation why you think this happened.

The report hare in the “report/ LuisAngelCabralGuzmanProject.csv” path, you can see that in some cases the TTC lidar is not calculated in correct way, this occurred if a few keypoints are detected, this happened when the Harris detector is used in combination with different descriptor, for all other detectors and descriptor combination the TTC using lidar sensor are very similar.

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.

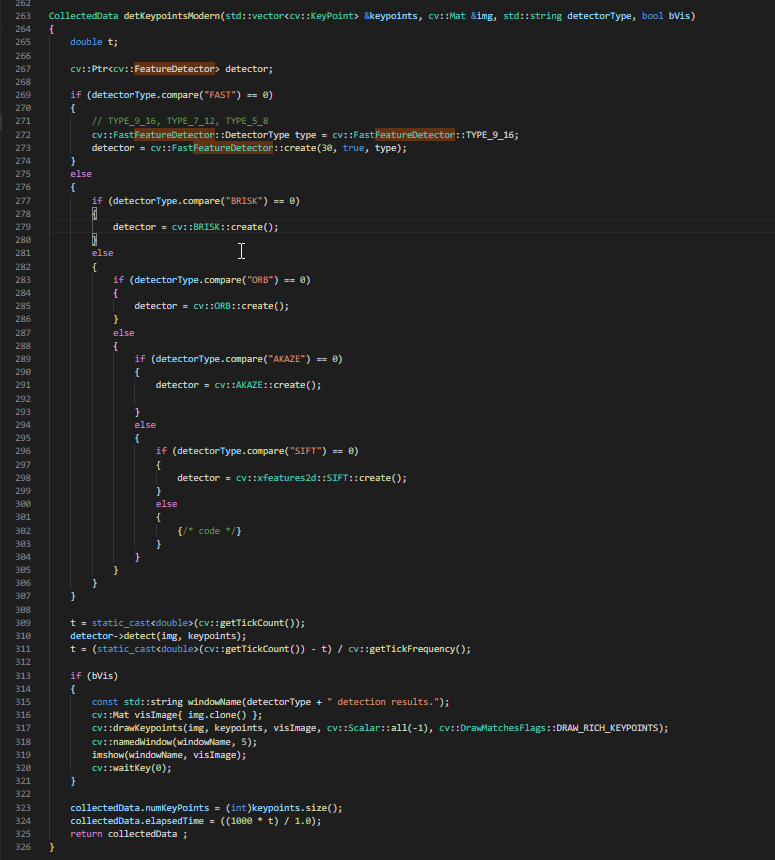
elative to each other

to know the From 249 to 314 line the keypoint maches are clustered and associated with bounding boxes in the images, from 264



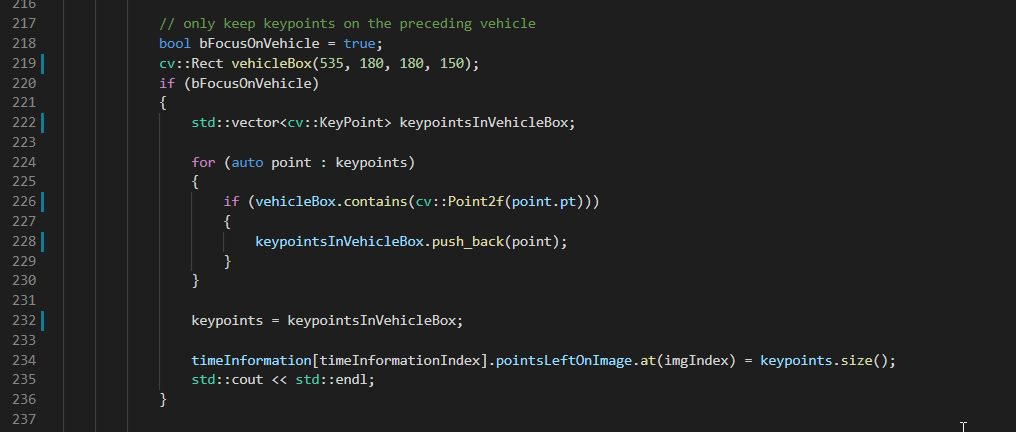
At line 201 the “detecotrType” variable is compared with HARRIS string after that the “detKeypointsHarris” function is called, it “detecotrType” is not equal to HARRIS string at line 207 “detKeypointsModers” function is called, inside of this function other if-else conditions compare the “detecotrType” variable with FAST, BRISK, ORB, AKAZE, and SIFT string in order to execute the proper detector, “detKeypointsModers” function is in “matching2D\_Student.cpp” file form line 263 to 326.

In “detKeypointsModers” method in the file “matching2D\_Student.cpp” from line 269 to 296 the detector is selected according to “detectorType”, after that at line 310 the “detector->detect” method is executed with the grayscale image as input and in the “keypoints” vector the detected keypoints are stored, in the next image you can see the “detKeypointsModers” method.



MP.3 Keypoint Removal: Remove all keypoints outside of a pre-defined rectangle and only use the keypoints within the rectangle for further processing.

In “MidTermProject\_Camera\_Student.ccp” at line 219 a bounding box named “vehicleBox“ is created that is used in the “if condition” at line 226 witch extracted all keypoints founded within the bounding box defined in “vehicleBox”, all keypoint within the “vehicleBox” bounding box are stored in the “keypointsInVehicleBox” auxiliary vector if the “vehicleBox.contains” method returns a true value after checks if the point is within the boundary.When the “for loop” from line 224 to 230 finishes the “keypoints” vector is assigned with all keypoints stored in “keypointsInVehicleBox” vector. You can see the code in the next image:



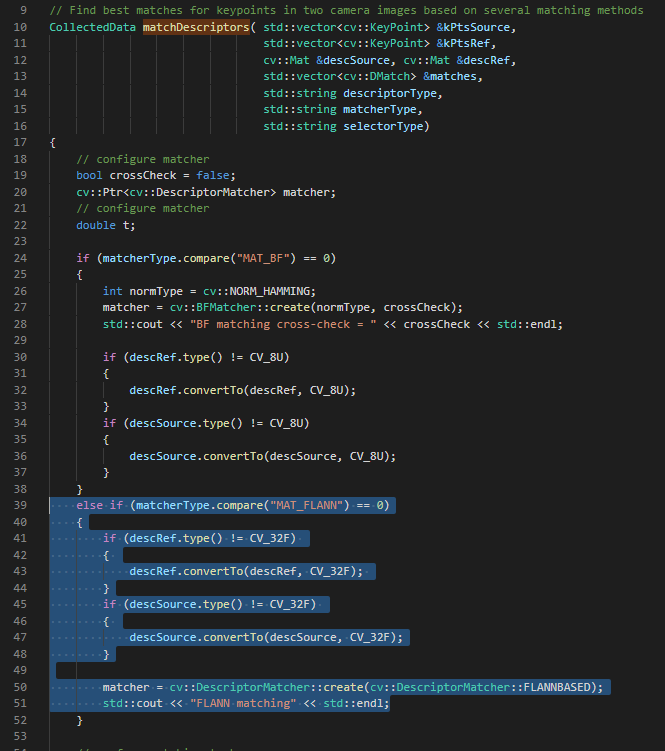
MP.4 Keypoint Descriptors: Implement descriptors BRIEF, ORB, FREAK, AKAZE and SIFT and make them selectable by setting a string accordingly.

In “matching2D\_Student.cpp” file from line 89 to 136 a sequence of “if-else” conditions compare the “descriptorType” variable with BRIEF, ORB, FREAK, AKAZE, and SIFT strings, when the “descriptorType.compare” method return a true value the generic extractor variable is assigned with the appropriate descriptor, and the compute method is called in the line 140.



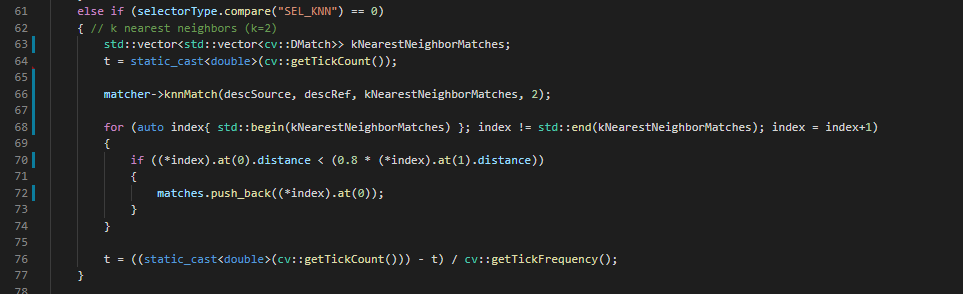
MP.5 Descriptor Matching: Implement FLANN matching as well as k-nearest neighbor selection. Both methods must be selectable using the respective strings in the main function.

In “matching2D\_Student.cpp” file from line 39 to 52 the FLANN matching is implemented, in the line 39, if the “matcherType.compare” method return a true value after compere with “MAT\_FLANN” string, two if conditions are used to convert the descriptor matrices to CV\_32F type if they are not CV\_32F type to avoid an OpenCV, after that the “matcher” variable of “DescriptorMatcher” data type is assigned a pointer to a descriptor matcher constructed with a FLANNBASED type.



MP.6 Descriptor Distance Ratio: Use the K-Nearest-Neighbor matching to implement the descriptor distance ratio test, which looks at the ratio of best vs. second-best match to decide whether to keep an associated pair of keypoints.

In “matching2D\_Student.cpp” file the K-Nearest-Neighbor selection is implemented, in line 68 a vector of “DMatch” type named “kNearestNeighborMatches” is used to store the matches from calling “matcher->knnMatch”, using a value of 2 for k, after that the descriptor distance ratio test is performed to 0.8 in each match in the “kNearestNeighborMatches” vector, for each point falling within the threshold distance is copied to the matches vector.



MP.7 Performance Evaluation 1: 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.

In line 234 in the “MidTermProject\_Camera\_Student.ccp” file the number of keypoints on the preceding vehicle are stored, the number of keypoints found on the vehicle is equal to the size of “keypointsInVehicleBox“ vector. At line 234 the number of keypoint in the vehicle are stored to write them in the CSV file. All detectors are implemented using the “for cycle” in line 145.

MP.8 Performance Evaluation 2: Count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors. In the matching step, the BF approach is used with the descriptor distance ratio set to 0.8.

In the “matching2D\_Student.cpp” file the size of “maches” vector at line 79 is the keypoints number using the distance ratio of 0.8, the “maches” size vector is stored to in the line 79 to write it in the CSV file. All detectors and descriptor are implemented using the “for cycle” in line 145 in “MidTermProject\_Camera\_Student.ccp” file.

MP.9 Performance Evaluation 3: Log the time it takes for keypoint detection and descriptor extraction. The results must be entered into a spreadsheet and based on this data, the TOP3 detector / descriptor combinations must be recommended as the best choice for our purpose of detecting keypoints on vehicles.

My top3 detector/descriptor are:

1. Fast detector / Brief descriptor
2. Fast detector / ORB descriptor
3. Fast detector / Brisk descriptor

These top 3 are in base of detector time and descriptor time that for all cases is not more than 3 secods, the high speed is important to real application, other important point is the matched keyPoints number, for those detector/ descriptor combination the matched points are around 100, I think they are enough, and the matcher time is close to 1 second., the CSV file are in the report folder.