Camera Based 2D Feature Tracking

Meets Specifications

Congratulations! This was an amazing submission. You could successfully load images, setup data structures and put everything into a ring buffer to optimize memory load. Next, you integrate several keypoint detectors such as HARRIS, FAST, BRISK and SIFT and compared them with regard to the number of keypoints and speed. You then performed the descriptor extraction and matching using brute force and also the FLANN approach. Lastly, you tested the various algorithms in different combinations and compare them with some performance measures. All requirements are correctly addressed. The efforts are appreciated, please carry on as I look forward to seeing future submissions from you.

I enjoyed reading through your source code and README. I've left some comments on your C++ style and how to take your code to the next level.

All the best and happy learning!

Extra Material

Check out the following to supplement your learning:
Cameras in Processing (2D and 3D)
How to Detect and Track Objects with OpenCV

Mid-Term Report

✓ **Provide a Writeup / README that includes all the rubric points a you addressed each one. You can submit your writeup as markd pdf.**

Well done as you have provided your project writeup in your subm this includes all the rubric points and how you addressed each one

Extra Tips

The links on README documents below can help.
How to write a great README
Suggestions on making a good README
About READMEs
How to put images in a README file
Make a README

Data Buffer

✓ **Implement a vector for dataBuffer objects whose size does not exceed a limit (e.g. 2 elements). This can be achieved by pushing in new elements on one end and removing elements on the other end.**

Well done! You have implemented a vector for `DataFrame` objects whose size does not exceed the limit. You created a data buffer with a size of 2.

Extra Resources

These resources should supplement your knowledge on data buffer implementation
Vector Erase
Simple Circular Buffer in C++

Keypoints

✓ **Implement detectors HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT and make them selectable by setting a string accordingly.**

Great job, you have implemented detectors HARRIS, FAST, BRISK, ORB, AKAZE, and SIFT!

Suggestions

A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB and BRISK
Harris Corner Detection
Fast Algorithm for Corner Detection
BRISK Algorithm

✓ **Remove all keypoints outside of a pre-defined rectangle and only use the keypoints within the rectangle for further processing.**

Nice work! This does indeed remove keypoints. Instead of looping t
the container manually and placing new keypoints in a new contair
used an erase-remove idiom to remove values in a container that r
certain criteria (https://stackoverflow.com/questions/39019806/usi
remove-if-idiom).

An alternative way is to use `cv::KeyPointsFilter::runByPixelsMask` which is admittedly not documented, but this method takes in a vector of keypoints and a mask that defines where valid keypoints should reside. You can define the mask to be the same size as the input image, and use the rectangular region defined by `vehicleRect` to remove keypoints that don't belong in that region.

```
cv::Mat mask = cv::Mat::zeros(imgGray.rows, imgGray.cols, CV_
8U); // all 0
cv::Rect vehicleRect(535, 180, 180, 150);
mask(vehicleRect) = 1;
cv::KeyPointsFilter::runByPixelsMask(keypoints, mask);
```

Descriptors

✓ **Implement descriptors BRIEF, ORB, FREAK, AKAZE and SIFT and make them selectable by setting a string accordingly.**

Great job, you have implemented detectors BRIEF, ORB, FREAK, AKAZE and SIFT!

Suggestions

A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB and BRISK
Harris Corner Detection
Fast Algorithm for Corner Detection
BRISK Algorithm

✓ **Implement FLANN matching as well as k-nearest neighbor selection. Both methods must be selectable using the respective strings in the main function.**

Amazing! You have implemented FLANN as well as K-nearest neighbor selection.

Extra resources

- You can go through this article to know more about FLANN Matching
- Examples of FLANN based matching
- KNN for Machine Learning

✓ **Use the K-Nearest-Neighbor matching to implement the descript distance ratio test, which looks at the ratio of best vs. second-be to decide whether to keep an associated pair of keypoints.**

Excellent work! This is implemented as described in your writeup. Y used the k nearest neighbor matching with K = 2 and minimum descriptor

Rate this review

START

distance of 0.8 to find the best match. Good job! I've left a comment on how to use range-based `for` loops which you should strive to use more often.

Extra Material

KNN Algorithm - Finding Nearest Neighbours
Feature Matching - OpenCV Tutorials

Performance

✓ **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.**

Well done! You have shown the count of the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors.

✓ **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.**

Excellent work with implementing the ratio test! Have a look at this discussion on Stack Overflow on how the ratio test actually works and why it's used as a popular outlier technique: https://stackoverflow.com/questions/51197091/how-does-the-lowes-ratio-test-work

✓ **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.**

A concise yet detailed discussion is provided for all the images. Nice job with the recommended choice for our purpose of detecting keypoints on vehicles.

- Great work explaining your top 3 choices. If we are going based upon speed, these are some great choices! I recommend considering what other factors and combinations could be used for other top 3 choices! Also, as a general note, this code will be used again later in the course, where you will build off of this project. Make sure to save this project for later!
- As a general recommendation, it's always a great idea to record your frame-by-frame data as well. In this case, it wasn't necessarily required, however it does help find outliers and trouble spots easier!

6 CODE REVIEW COMMENTS ❯

RETURN TO PATH

Rate this review

START