LAB REPORT FOR CHRISTOPHE BOBDA'S GROUP

WEEK 3

Xianhan Li

INTRODUCTION

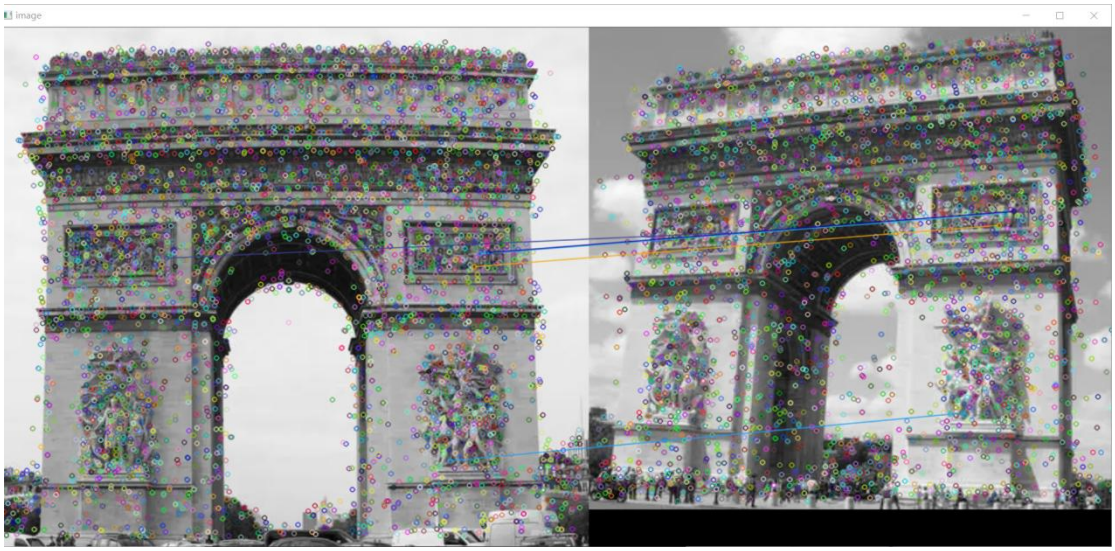
This report includes 3 tasks: The first one is about feature detection with SURF, I used SURF feature detector to detect keypoints and drawn them out on the images, and then chose 5 pairs of features detected and connected each pair; The second one is about pose estimation of chessboards, including detecting a chessboard in a new image and finding its distance from the camera; The last task is reading the paper ***Hybrid Camera Pose Estimation***, Writing a brief summary of the paper and trying to implement the method discussed in the paper.

TASK1

I read in the images and change them to grayscale images. And then, I built SURF objects using "cv2.xfeatures2d.SURF_create()", then I detected and compute the key points and descriptor using "sift.detectAndCompute". After, I used the "cv2.drawKeypoints" function to draw the extracted key points on a grayscale image, generating images with the key points "image_with_kp1" and "image_with_kp2" respectively. Then I used FLANN to match the key points and drew them out on one image like the task 2 in last week.

The codes of this task are in the file **"T3_task1.py"**.

Result of Implement:



TASK2

STEP1: Define a function "generate_chessboard_points(board_size, square_size)" to generate 3D coordinates of the chessboard corners in a given board size and square size.

STEP2: Load the two chessboard images named "chessboard01.jpg" and "chessboard02.jpg".

STEP3: Define chessboard size and square size.

STEP4: Use “cv2.findChessboardCorners()” to find the corners of the chessboard.

STEP5: Generate 3D coordinates of the chessboard corners using the “generate_chessboard_points()” function.

STEP6: Read camera intrinsic parameters from a YAML file named "camera_intrinsics.yml" using “cv2.FileStorage()”.

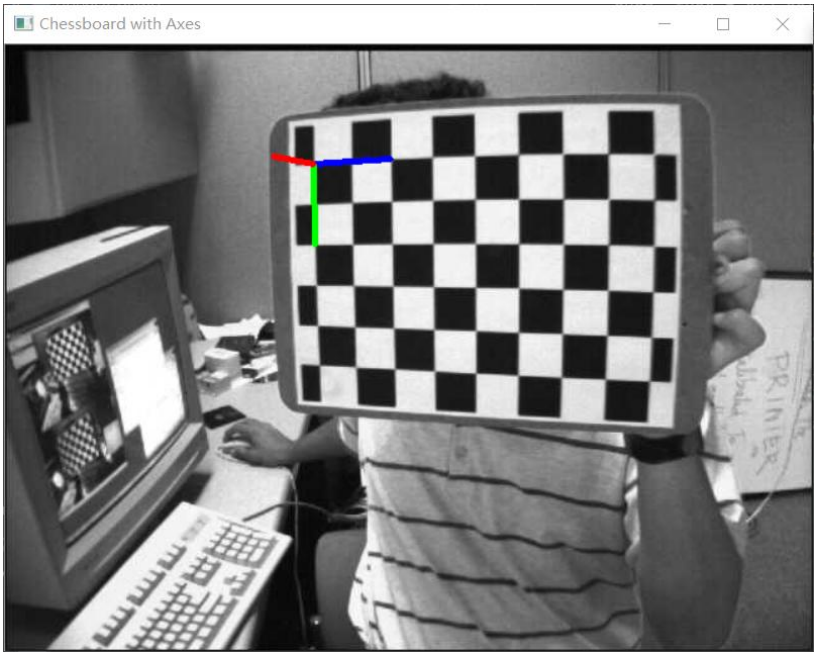
STEP7: Find chessboard pose using “cv2.solvePnP”.

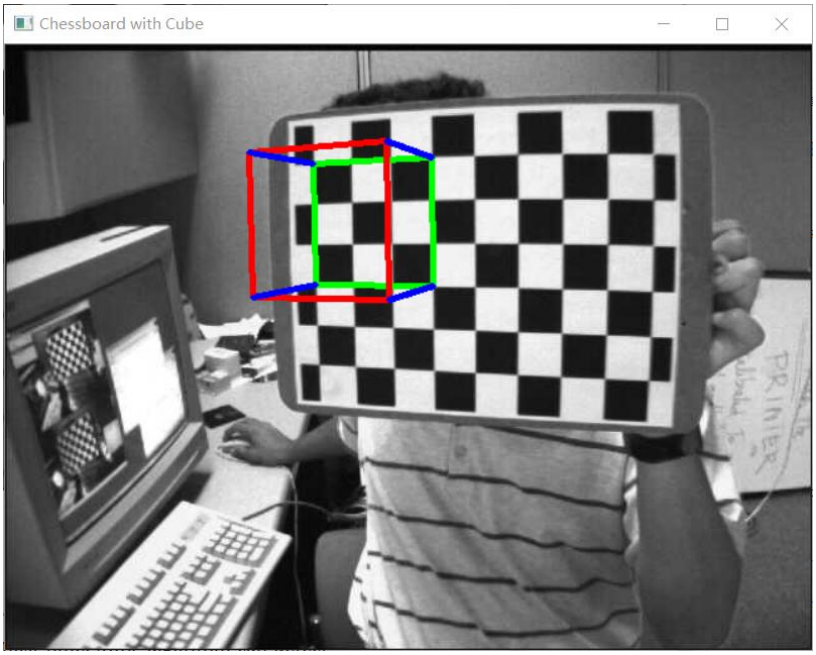
STEP8: Project 3D axes onto the chessboard, and draw the projected axes on the second chessboard.

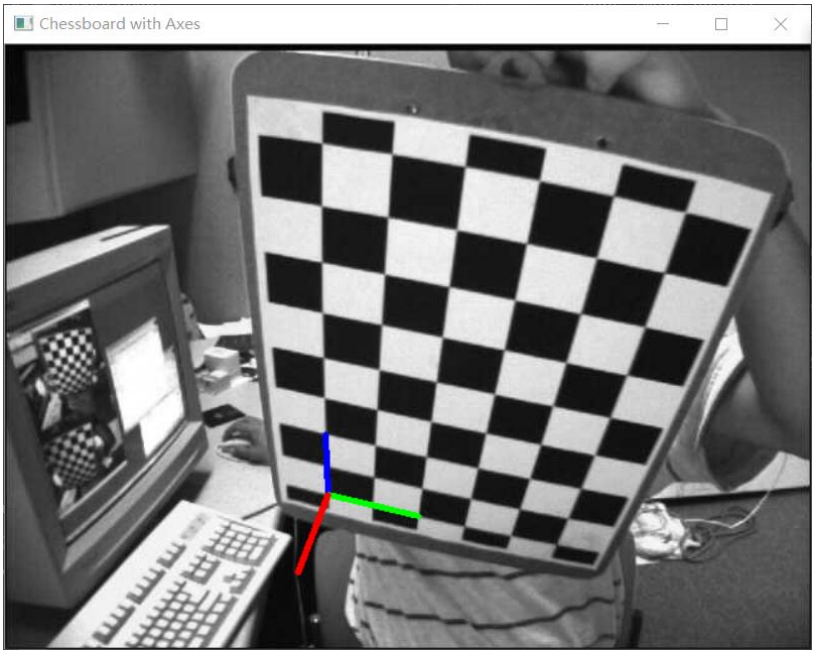
STEP9: Project a cube onto the chessboard, and define cube edge connections and colors for each edge, and then draw the projected cube edges on the second chessboard.

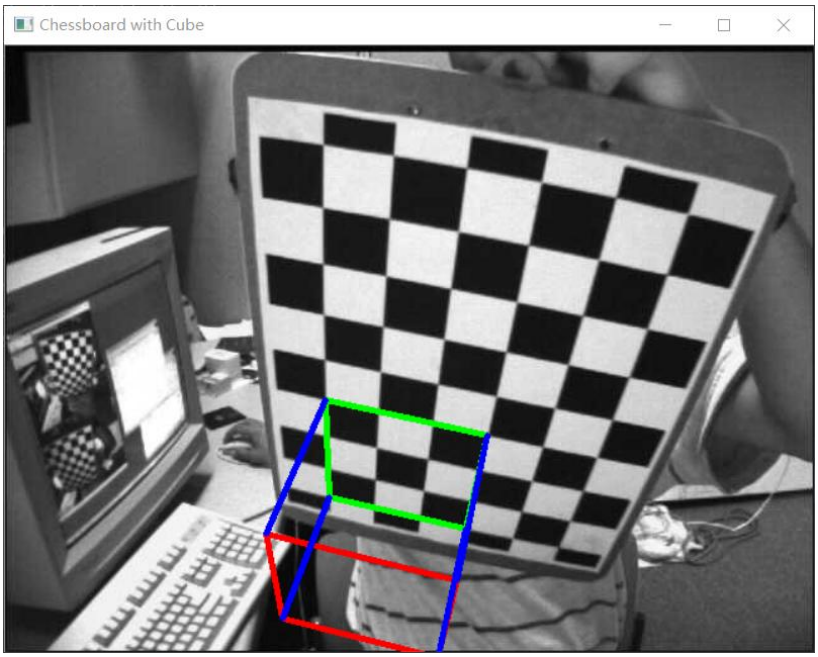
The codes of this task are in the file **“T3_task2.py”**.

Results of Implement:




Chessboard with Cube


Chessboard with Axes



Task3

Brief summary of the paper:

This paper solves the problem of camera pose estimation with respect to a Structure-from-Motion (SfM) model by leveraging both 2D-3D and 2D-2D correspondences. Traditional methods either focus on 2D-3D matches (structure-based pose estimation) or solely on 2D-2D matches (structure-less pose estimation). The authors propose a **novel RANSAC-based approach** that dynamically selects the best type of **solver** for each iteration. This can range from purely structure-based or structure-less solvers to a combination of hybrid solvers that use both types of matches. The paper introduces several **new hybrid minimal solvers**. Through experiments, the authors demonstrate that their approach achieves accuracy comparable to structure-less methods while maintaining the efficiency of structure-based methods.

Implement of the method:

RANSAC is the core part of the algorithm introduced in this paper, and its implementation consists of several steps:

1. Given a data set S, select the minimum number of samples required to build the model from it (the spatial line can be determined by at least two points, so the minimum number of samples is 2; the spatial plane can be determined by three non-collinear points, so the minimum number of samples is 3; when fitting a circle, the minimum number of samples is 3), and remember the data set as S1.

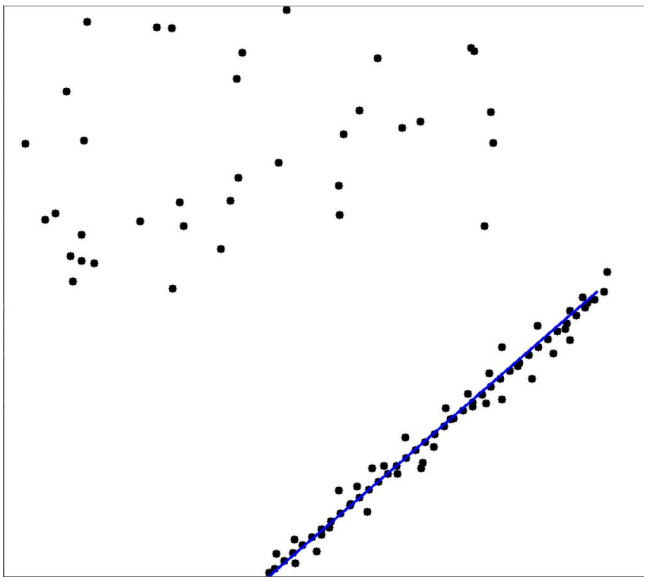
2. Compute a mathematical model M1 using the selected data set S1.

3. Use the calculated model M1 to test the remaining points in the data set. If the tested data point is within the allowable range of error, the data point is judged as an inlier; otherwise, it is judged as an outlier. The data set composed of all the internal points is recorded as S1*, and S1* is called the consistent set of S1.

4. Compare the number of "inside points" of the current model with the best previously released model, and record the maximum number of "inside points" when the model parameters and the number of "inside points".

5. Repeat steps 1-4 until the iteration ends or the current model is good enough; Each time a model is produced, it is either rejected because there are too few interior points, or it is chosen because it is better than the existing model.

Results of Implement:



Conclusion

In this week, I have got in touch with feature detection with SURF which I have implemented last week, and pose estimation of chessboards. It makes me have deeper impression on Computer Vision. But since it is a totally new area for me, I may make some mistakes in my assignments. Also, I have read a paper about camera pose estimation, which is very challenging for me. Although I have not totally understand all the experiments in the paper, I have known the principle of the core algorithm, RANSAC, and tried to implement it.