Name: Remaldeep Singh
UID: u1143744
Email: u1143744@utah.edu

This project is divided into three parts:

- CameraCalibration.
- Pose Estimation.
- Bag of Words.
- Vocabulary Tree.

The main.m file in the project directory is used to call all other functions. In this assignment I have calculated a lot of variables and stored them in **".mat" format files** to speed up the runtime.

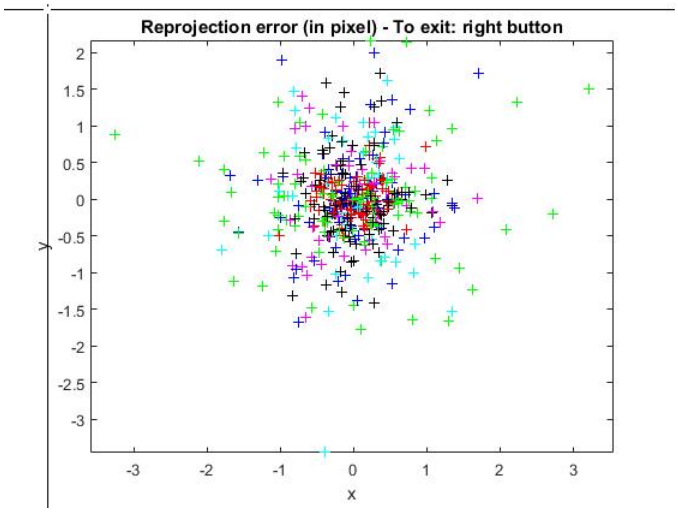
Camera Calibration

This part doesn't have code but includes running the calibration toolbox to get the camera calibration parameters. I used images provided by Professor Srikumar. The calibration reprojection error came out to **0.6639,0.6258**.

The calibration matrix after re-computation of corner points and re-calibration is:

1439.7	0	897.27
0	1442.4	475.39
0	0	1

Reprojection error figure:



Calibration results (with uncertainties):
Focal Length: fc = [1439.67241 1442.41621] +/- [23.26253 23.44372]
Principal point: cc = [897.27049 475.39406] +/- [23.02318 17.23369]
Skew: alpha_c = [0.00000] +/- [0.00000] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion: kc = [0.05559 -0.09180 -0.00919 -0.00590 0.00000] +/- [0.02357 0.04490 0.00363 0.00412 0.00000]
Pixel error: err = [0.63386 0.62577]

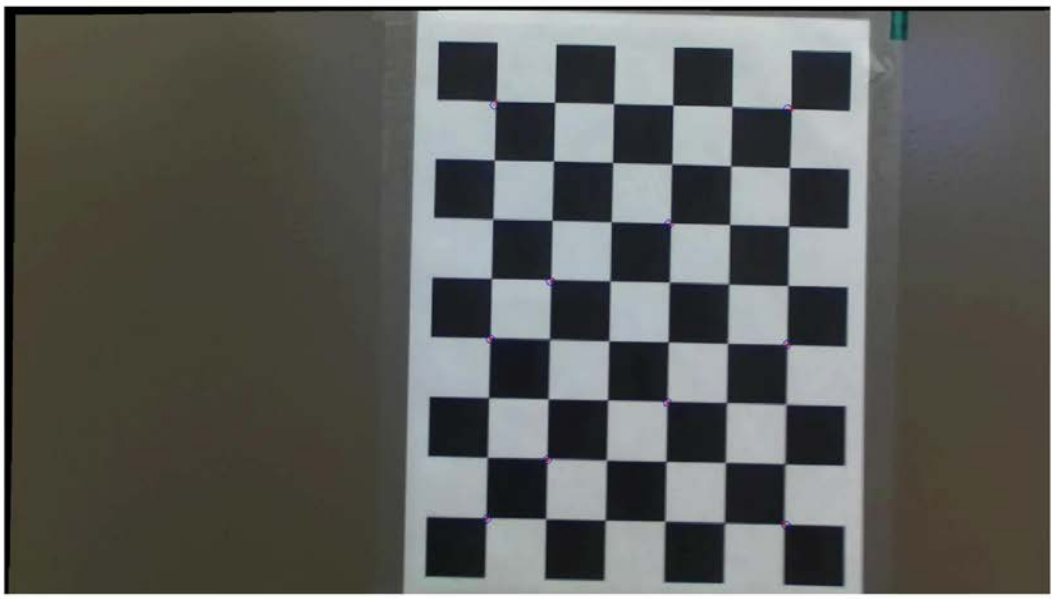
Pose Estimation

This part is under Step 1 of main.m file.

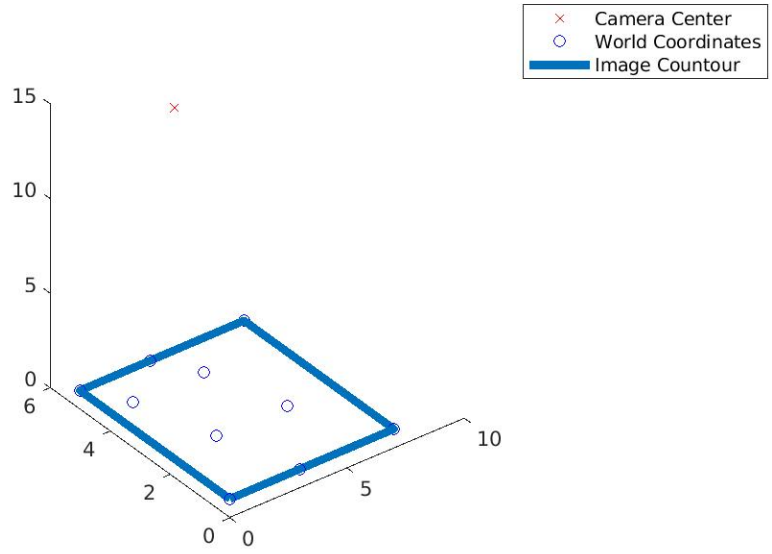
For this problem I ran ginput on a sample undistorted image and selected 10 points. The image coordinates of these points are present in **q_i** matrix and their world coordinates are present in **Q_m_i**.

The calculate_Permutations method returns $^{10}C_3$ combinations of the 10 points. Rather doing RANSAC on these combination I selected a few of them are results came out good. Hence the value of i is 54 which give on possible combination of 3 points. **data_g** has the combinations and we select row 54 from it. q1, q2, q3 are indices to Q_m_i and q_i for the image points. After checking that the points don't lie on the line we calculate Q_c_i which is product of inverse of calibration matrix with the image coordinates. X_i, Y_i, Z_i are calculated consequently from these Q_c_i's. With the lambda variables as l1, l2, l3 the 3 equations are give to the vpasolver and 8 results are returned. After filtering only the real and positive results for lambda's, we calculate the calibration matrix using the Register3DPointsQuaternion method with Q_m_i's and calculated Q_c_i's. The rotation (**rot_m**) and translation (**trans_m**) matrices are calculated form this calibration matrix.

Using these rotation and translation matrices we re-project the image points q_i and add the error in **RPE**. Out of all the possible lambda solutions we take the on that has the least reprojection error. After finding out the minimum reprojection error, the translation and rotation matrices are saved as **trans_m_min** and **rot_m_min.** Later on, these 2 matrices are used to reproject the points to the image coordinate frame and the output is shown in 'Ouputs/pose_estimation.1.jpg'.



The chosen lambda's, translation and rotation matrices are visualized below with the world coordinates.



Bag of Words

This part is under Step 2 of main.m file. The method Clustering implements the bag of words algorithm. With **1000 clusters** I am getting an accuracy of **96%** on the dataset.

The **descriptors** variable contains all the sift descriptors of all the database images. The **meta_cell** variable is a cell 2D array where each row has the image name, the image descriptor and the image cluster vector. Using SIFT I calculated the descriptors for all the database images and stored them in descriptors.mat file. The meta_cells are stored in meta_cell.mat file. And the results of kmeans clustering with 1000 clusters is stored in C.mat and idx.mat file. All these variables are loaded from the respective files on every run.

In order to make the cluster for an image, the method loops over each row of meta_cell and for each descriptor of that image check which cluster it belongs to. It then increments the value of that cluster in the **cluster_info** array. cluster_info array is the cluster vector for an image. This is sotred in 3rd column of the meta_cell matrix.

The query_cells.mat is the cell matrix containing descriptors for every query image rowise. After running the sift program on each query image, the descriptors are stored in query_cells.mat program. The method starts by forming a cluster vector for every query image by calculating the pdist2 (cosine distance) between every descriptor of the image and the cluster centers "C" and then taking minimum distance value to get the minimum distanced cluster. The cluster_info for the cluster is then incremented by 1. Once all the descriptors are done **knnsearch** algorithm is run for the query cluster vector and the img_clusters(which contains the cluster vector for every image in the database). Top 5 results are retained and checked if the retrieved results match with actual results from the database. With even 1 match we treat the query as positive or otherwise negative. I was able to achieve 96% accuracy on this dataset.

Vocabulary Tree

This part is under Step 3 of main.m file. The method vocab_tree() implements the vocabulary tree algorithm. I will be reusing the descriptors.mat and meta_cell.mat variables from the bag of words in this part.

Create_nodes method takes in root, descriptors, levels and branches as input and returns a tree. Inside the method if the level becomes 0 then it returns -1 otherwise it does k means clustering. The root cell array has 6 elements, where 1st element is a vector all the centers of the clusters and the next 5 elements are another cell arrays of the same type as root. Create_nodes is called recursively as many times as there are branches with new root, new descriptors, level decremented by 1and branches. The new_descriptors are derived from the clustered descriptors whose indices match with branch index. The return value of create_nodes is inserted the correct position in the root cell array.

The traverse_tree method takes in root, descriptors and level as input and returns the path of the tree for that descriptor. If the level becomes 0 the method returns -1. The method starts by calculating pdist2 (L2 i.e eucledian distance) of the descriptor from the 5 centers of this node and selects the center with the minimum distance. This follows a recursive call to the traverse_tree with new_root as the cell array with minimum index, the same descriptor and level decremented by 1. The return value of traverse_tree is appended with the minimum index at every level and the final return has the path taken by the descriptor in the tree. There is an extra -1 at the end of this final return variable which is later removed in the main code.

The method total_nodes takes in branches and levels as input and returns the total number of nodes in the tree as ouput.

For each of the row in meta_cell i.e for each image in the database descriptors are extracted and for each descriptor the tree is traversed. Since the output of tree traversal, **wordV,** does not have an offset with respect to total number of nodes present at that level, the **wordV_N** variable stores the actual offset outputs. The offset is increased in the order of branches every time. The array **cluster_info** for an image contains the histogram for every cluster. Its length is equal to the return value of the method total_nodes. Each cluster present in the descriptor of an image increments the corresponding cluster index in cluster_info. Cluster_info is then inserted in the third column of meta_cell. The tree “root” and cell matrix “meta_cell” is stored as root.mat and meta_cell_tree.mat for further reuse.

All the image clusters for the database images are consolidated in the matrix img_clusters. The processing of query image is similar to database images. query_cells.mat has the descriptors for every query image. For each query image all the descriptors are extracted from the query_cells and cluster_info is made for this image using similar steps previously mentioned. Once we have the cluster_info knnserach is applied on the cluster_info and the img_clusters with top 5 results retained. If we are able to retrieve atleast one of the right database images we mark this query as positive otherwise negative.

Outputs:
94% accuracy with 4 levels and 5 branches.
86% accuracy with 3 levels and 5 branches.
92% accuracy with 3 levels and 4 branches.
90% accuracy with 4 levels and 4 branches.
94% accuracy with 4 levels and 6 branches.

Although the above data feels a bit random but as we increase the levels and branches the accuracy seems to increase although this is a very small dataset to comment on right now.