CS 6320 Computer Vision

Homework 2 (Due Date – February 18th)

1. Download the Matlab calibration toolbox from the following page:

   http://www.vision.caltech.edu/bouguetj/calib_doc/

   Download the calibration pattern from the following link:

   http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/pattern.pdf

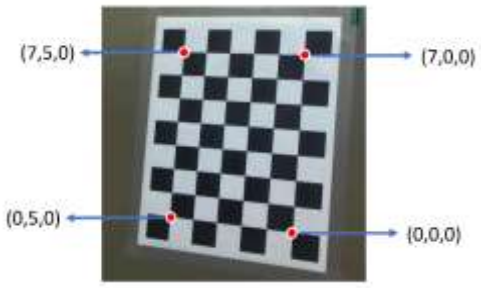
   Print the pattern and paste it on a planar glass wall or table. Note that each of the squares is of length 30 mm. The surface must be perfectly planar, and the glue should not cause too many distortions. Take 10-15 images of the calibration grid using your digital camera (after fixing the parameters to manual mode), and follow the instructions provided in the first calibration example from the Matlab calibration toolbox. You can also use the images provided in this assignment. However, we recommend that you use your images. If you capture your images, you need to make sure that the camera parameters do not change while calibrating the camera.

   http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html

   You can start the calibration program using "calib_gui" command. During the calibration, you will be asked to click the four corners for every calibration grid image. Try to click these points as accurate as possible. If necessary, use some mouse magnifier. In addition to intrinsic parameters such as focal length and principal point, we also have radial distortion parameters. This distortion is responsible for making a line look like a curve. All the parameters are computed by the toolbox. The radial distortion parameters are not part of the 3x3 calibration matrices. For any new image you capture from this camera, please remove distortions first using the command "undistort_sequence.m". Following this, you have a 3x3 calibration matrix K for every undistorted image from this camera. Show the calibration matrix and the associated reprojection error (some measure of how good the calibration is) that is produced by the calibration software. For high resolution images, you will get a more substantial reprojection error (maybe 2-3 pixels) since the error is calculated in pixels.

   [20 Points]

2. Using the calibration matrix obtained in the previous step, we will implement and test simple pose estimation module. For one of the calibration grid images, first, correct the distortions and manually click the pixel locations for ten 2D points in the image. For each of the 2D points, provide the 3D points manually as shown below:



We show four 2D points and the corresponding 3D points (specified in the world coordinate frame). The idea of giving manual 3D coordinates in world coordinate frame is straightforward. We need to give the points in some convenient coordinate frame. To do that, you just need to know the dimensions of the 3D object. Knowing the dimensions up to a scale is sufficient. For example, in the above checkboard example, the points in the world coordinate frame is given in some metric, not necessarily centimeters or meters.

Using a subset of 3 points, compute the pose by solving the three quadratic equations. You will get eight solutions. Choose only solutions where the 3D points on the projection rays are real and have positive 'z' values. Note that the pose computation also requires Horn's quaternion method for computing the motion between the 3D points in the camera coordinate frame and the world coordinate frame. The code for motion estimation between two sets of 3D points using Horn's method is provided with the assignment (Register3DPointsQuaternion.m, Register3DPointsQuaternion.py). Each solution corresponds to a pose (R, T). For each of the solutions, try checking the reprojection error from all the ten 3D points. Choose the pose that gives the lowest mean reprojection error. Visualize the results in 3D. Note that you can use "plot3d" to visualized 3D points in Matlab.

We give a simple example to use "solve" command in Matlab. We would like to solve 3 quadratic equations {a^2 * x^2 = b, b^2 * y^2 = c, c^2 * z^2 = a * x +b * y} in 3 variables (x,y,z):

```
>> syms x y z

>> a = 2; b =3; c =4; d =5;

>> solve(a^2 * x^2 == b,b^2 * y^2 == c,c^2 * z^2 == a * x +b * y);

>> sols = solve(a^2 * x^2 == b,b^2 * y^2 == c,c^2 * z^2 == a * x +b * y);

>> x_ = double(sols.x);

>> y_ = double(sols.z);

>> z_ = double(sols.z);
```

We obtain eight solutions each for x, y, and z. The solutions are given as vectors in x_, y_, and z_. You can use "isreal" command to identify imaginary solutions and discard them.

We give a simple example to use "solve_poly_system" command in Python. We would like to solve 3 quadratic equations {a^2 * x^2 = b, b^2 * y^2 = c, c^2 * z^2 = a * x +b * y} in 3 variables (x,y,z):

```
from sympy import solve_poly_system

from sympy.abc import x,y,z

from numpy import isreal

a = 2

b = 3

c = 4

d = 5

sols = solve_poly_system([a**2 * x**2 - b, b**2 * y**2 - c, c**2 * z**2 - a *x -b * y], x, y, z)

# there are usually 8 solutions by solving the above system.

for i in range(0,8):

    print("soln -",i,"(x=",sols[i][0].evalf(),"y=",sols[i][1].evalf(),"z=",sols[i][2].evalf())
```

Steps in Pose Estimation:

Before you start the pose estimation, make sure that you have distortion corrected image (after using undistort_sequence command on an image) and the calibration matrix K.

1. Using "ginput" click a set of 10 points given by $q_i, i = \{1, \dots, 10\}$ for which you know the 3D locations in the world coordinates given by $Q_i^m, i = \{1, \dots, 10\}$.
2. Select a set of 3 2D points (q_1, q_2, q_3) does not lie on a straight line.
3. The corresponding 3D points on the camera coordinate frame are given by: $Q_i^c = \lambda_i K^{-1} q_i, i = \{1,2,3\}$.
4. We can rewrite this in a simpler form: $Q_i^c = \lambda_i \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}, i = \{1,2,3\}$. Note that $\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}$ is completely known.
5. Find the pairwise distances from the points in the world coordinates:

$$d_{12} = norm(Q_1^m - Q_2^m);\ d_{23} = norm(Q_2^m - Q_3^m); d_{31} = norm(Q_1^m - Q_3^m);$$

6. Using pairwise distances, we get obtain eight solutions for $(\lambda_1, \lambda_2, \lambda_3)$ using the 3 quadratic equations that we show in the lecture notes.

$$(\lambda_1 X_1 - \lambda_2 X_2)^2 + (\lambda_1 Y_1 - \lambda_2 Y_2)^2 + (\lambda_1 Z_1 - \lambda_2 Z_2)^2 = d_{12}^2$$
$$(\lambda_2 X_2 - \lambda_3 X_3)^2 + (\lambda_2 Y_2 - \lambda_3 Y_3)^2 + (\lambda_2 Z_2 - \lambda_3 Z_3)^2 = d_{23}^2$$
$$(\lambda_3 X_3 - \lambda_1 X_1)^2 + (\lambda_3 Y_3 - \lambda_1 Y_1)^2 + (\lambda_3 Z_3 - \lambda_1 Z_1)^2 = d_{31}^2$$

7. For each of the eight solutions (that are not imaginary and negative), compute Q_i^c. Compute rotation and translation using

$FinalTrans$ =`Register3DPointsQuaternion`$([Q_1^m\ Q_2^m\ Q_3^m], [Q_1^c\ Q_2^c\ Q_3^c])$.

The transformation matrix "FinalTrans" is a 4x4 matrix.

$$R = FinalTrans(1{:}3,1{:}3); T = -R^T\ FinalTrans(1{:}3,4)$$

8. Now compute the mean reprojection error using [R,T] for all the 10 2D pixels $q_i, i = \{1, \dots, 10\}$ by projecting the world points $Q_i^m, i = \{1, \dots, 10\}$ using the computed [R,T] and known values of K matrix. Find the lambdas that produces the smallest mean reprojection error.

$$Reprojection: q_i' = \begin{bmatrix} x_i'/w \\ y_i'/w \\ 1 \end{bmatrix} \sim \begin{bmatrix} x_i' \\ y_i' \\ w \end{bmatrix} = K(RQ_i^m - RT), i = \{1,2, \dots 10\}$$

$$\text{Mean RPE} = \frac{1}{10}\sum_1^{10} \sqrt{(q_{ix} - q_{ix}')^2 + (q_{iy} - q_{iy}')^2}$$

[If you enjoy the pose estimation experiment, you can think about your final course project in some augmented reality application.]

[40 Points]

3. The goal is to obtain image-based localization. The dataset consists of two folders: query and database. The query folder has 50 images, and the database folder consists of 200 images. For every image A.png (A is some integer) in the query folder, you have four images (A_1.png, A_2.png, A_3.png, and A_4.png) that are less than 10 meters from the query image A.png. You will develop two slightly different localization algorithms, one using Bag of Words (BOW) and the other using the vocabulary tree. Both BOW and vocabulary tree require some clustering algorithm to partition the set of descriptors from the database images into certain number of clusters. In Matlab, K-means clustering can be done with a simple command "kmeans".

   We will use a simple evaluation procedure to compute the accuracy. For every image A.png in the query folder, you will retrieve the top 5 closest images in the database folder using BOW and Vocabulary tree. If at least one of the five retrieved images is from the set (A_1.png, A_2.png, A_3.png, and A_4.png), then you treat the retrieval as a success.

Database: Four of the images from the database folder are given above.

Database: Four of the images from the query folder are given above.

The accuracy should be evaluated based on the total number of successful retrievals divided by the total number of query images. You will not be required to implement SIFT for this assignment. You can use the code that is available here:

http://www.cs.ubc.ca/~lowe/keypoints/

BOW parameter settings: Extract n (say 1000 or 2000) keypoints from every image in the database. Consider having K=1000 clusters.

Vocabulary tree parameters: You can have a branch factor 4 or 5 and consider L = 3 or 4 levels.

You will be required to play with different parameter settings and study the accuracy. In the vocabulary tree, you can consider using L1 or L2 distances for measuring the distance between query and database images.

[If you enjoy the image-based localization, there are many related ideas that you can do for the final course project.]

[40 Points]