Shubham Suresh Patil
Andrew ID: shubhamp
shubhamp@andrew.cmu.edu

Geometry Based Methods in Vision
Assignment-2

2019/10/25

1 Theory

1.1 Join & Meet

1.1.1

Given that we are in the P_2 space, the cross product of two points is a line while the cross product of two lines is a point in P_2.

so,

$$P_1 V P_2 = P_1 \mathrm{x} P_2$$

and

$$l_1 \wedge l_2 = l_1 \mathrm{x} l_2$$

1.1.2

The dual of

$$\mathrm{L} = P_1 V P_2$$

is a pencil of planes passing through points P_1 and P_2 in P^3 space as L is defined as the joint of P_1 and P_2. We will need only two planes to define the dual of L.

1.1.3

let Π be the plane and M_{Π}^{+} be the projection of point p_1 to P_1, p_2 to P_2 and p_3 to P_3 respectively. Now we get:

$$\begin{aligned} P_1 &= M_{\Pi}^{+}\, p_1 \\ P_2 &= M_{\Pi}^{+}\, p_2 \\ P_3 &= M_{\Pi}^{+}\, p_3 \end{aligned}$$

The projections of points on plane Π is given as follows:

$$\Pi = P_1 V P_2 V P_3$$

This join of $P_1 V P_2 V P_3$ represents the plane Π. This can be thought as the joint space of span of line defined by $P_1 V P_2$ and point P_3 which defines a plane. Note: Ensure that point P_1, P_2, P_3 are distinct and not collinear. Join of three image coordinates is a plane.

1.2 Plane+Parallax

1.2.1

In the given question, Hp_1 is the parallax point.

We know the relation between Fundamental Matrix F and points p_1 and p_2 as:

$$p_2^T F p_1 = 0 \tag{1}$$
$$p_2^T [e^{'}]_x H p_1 = 0 \tag{2}$$

Here, $[e]_x$ is skew-symmetric matrix. From the (1) and (2) we get we get,

$$p_2^T([e^{'}]\text{x}H)p_1 = 0$$
$$p_2^T([e^{'}]\text{x}Hp_1) = 0$$

Thus, p_2, Hp_1 and $e^{'}$ are aligned.

1.2.2

From the previous question we proved p_2, Hp_1 and $e^{'}$ are aligned. If 3 points lie on the same line i.e. they are collinear, any one point can be expressed as a linear combination of other two points.
i.e.

$$p_{n3} = \lambda_1 p_{n1} + \lambda_2 p_{n2}$$

Here,

$$\lambda_1 p_{n1} = Hp_1$$
$$\lambda_2 p_{n2} = re'$$

1.2.3

Given:

$$M_1 P = p_1$$
$$M_2 P = p_2$$
$$(p_2)^T E_{23} p_3 = 0$$
$$M_1 P_\pi = p_1$$
$$Hp1 = p_3$$

Here $E_i j$ represents the Essential Matrix between cameras i and j.

$$p2^T E_2 3 H p_1 = 0$$
$$(p2)^T E_{12} p1 = 0$$
$$E_{12} = E_{23} H$$
$$p_2^T [t]_x R p_1 = 0$$

also,

$$H = R - (t * n^T) \div d. \tag{3}$$

We can rearrange (3) so that we can get

$$R = H + (tn^T) \div d$$

and then we can substitute it in above equation:

$$p_2^T [t]_\times (H + (t * n^T)/d) p_1 = 0$$

Here we are cancelling t and t, so that we are left with: (p2) T [t] Hp1 Now we can put back in p3: (p2).T[t] p3 = 0 Hence these 2 points are just related by pure translations.

1.2.4

The translation between points 1 and 2 is - t as found in the previous question. The translation between image 1 and image 2 can be found as follows:

$$M_1 = \begin{pmatrix} I & 0 \end{pmatrix}$$
$$M_2 = \begin{pmatrix} R & t \end{pmatrix}$$
$$M_3 = \begin{pmatrix} R & t^{'} \end{pmatrix}$$

Correspondingly we get,

$$M_1 = M_1^{'} Q$$
$$M_2 = M_2^{'} Q$$
$$M_3 = M_3^{'} Q$$
$$M_3^{'} = \begin{pmatrix} I - t \end{pmatrix}$$
$$Q = \begin{pmatrix} R & t \\ s & 1 \end{pmatrix}$$

substituting the value of Q in M_3 we get:

$$M_3 = \begin{pmatrix} I - t \end{pmatrix} \begin{pmatrix} R & t \\ s & 1 \end{pmatrix}$$
$$M_3 = \begin{pmatrix} R - ts & t - t \end{pmatrix}$$
$$M_3 = \begin{pmatrix} R - ts & 0 \end{pmatrix}$$

So, the translation is zero in this case.

1.3 Calibration of a moving set of camera

1.3.1

Given that,

$$P_0^{'} = QP^{'} \tag{4}$$
$$P^{'} = HP \tag{5}$$

substituting the (4) in (5), we get:

$$P_0^{'} = QP^{'}$$
$$P_0^{'} = QHP$$

It is given that,

$$P_0 = QP$$
$$P = Q^{-1}P_0$$

substituting the value of P in the previous equation:

$$P_0^{'} = QHP$$
$$P_0^{'} = QHQ^{-1}P_0$$

QHQ^{-1} is equal to H_0. Let v be the eigen vector of H_0

$$H_0 v = \lambda v$$
$$QHQ^{-1}v = \lambda v$$
$$HQ^{-1}v = Q^{-1}\lambda v$$
$$HQ^{-1}v = \lambda Q^{-1}v$$

Thus, $Q^{-1}v$ is an eigen vector of H with the same eigenvalues.

1.3.2

The dual of a transformation matrix is equal to the inverse transpose of the tranformation matrix.

$$T* = T^{-T}$$

$$T^{-T} = \begin{pmatrix} R^T & 0 & t^T 1 \end{pmatrix}$$

Given the equation that $T * pi_{infinity} = (lambda) pi_{infinity}$ we can put in the above to find:

$$\begin{pmatrix} R^T & 0 & t^T 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

The transformation only gets multiplied by 0's so it goes away and we are left with:

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \lambda * \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

which means the plane at infinity is an eigen vector.

1.3.3

Here

$$H^{-T}\Pi_\infty = \lambda_\infty \Pi_\infty$$

From the proofs above we can substitute H

$$\begin{aligned}(Q^{-1}H_oQ)^{-1}\Pi_\infty &= \lambda\Pi_\infty \\ QH_o{}^{-T}Q^{-T}\Pi_\infty &= \lambda\Pi_\infty \\ H_o{}^{-T}Q^{-T}\Pi_\infty &= Q^{-1}\lambda\Pi_\infty\end{aligned}$$

Because Q is a metric rectification matrix, we can now tell that $\Pi_\infty = \lambda \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$ and hence:

$$H_o{}^{-T}\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \lambda\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Since H_o is in Euclidean space,

$$T*\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \lambda\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Hence the plane at infinity is the eigen vector of H^T with real eigen vectors.

1.4 The special matrix

1.4.1

Given three vectors u, v, w, the double cross product is:

$$u \times (v \times w) = (u^T w)v - (u^T v)w \tag{6}$$

Substituting u = p, v = p and w = q in (6), we get,

$$p \times (p \times q) = (p^T q)p - (p^T p)q$$
$$[p]_x[p]_x q = (p^T q)p - (||p^2||)q$$

By Kronecker product,

$$[p]_x[p]_x q = (p \otimes p)q - (||p^2||)q \tag{7}$$
$$[p]_x^2 = (p \otimes p) - (||p^2||) \tag{8}$$

The equation (8) refer to the final solution.

1.4.2

Given that,

$$S = [e]_x F \tag{9}$$

Multiplying both the side in (9) by $[e]_x$, we get,

$$[e]_x S = [e]_x[e]_x F$$

From (8) we get,

$$[e]_x S = (e \otimes e)F - (||e^2||)F \tag{10}$$

$(e \otimes e)F$ = 0, so equation (10) becomes,

$$[e]_x S = -(||e^2||)F$$

which means $[e]_x S = (uptoscale)F$

1.4.3

In the above question, we had

$$S = [e]_x F$$

Here $[e]_x$ is a skew-symmetric matrix with rank 2. The fundamental matrix F also has a rank of 2. The product of two rank 2 matrices is another matrix of rank 2 or we can say det(S) = 0.

2 Let us do some implementation!

2.1 Vanishing Point Detection

2.1.1 Description of Implementation

The following is the algorithm:

1. Please refer to the matlab file named $question2_1.m$ for the implementation.
2. I have created a data loader wherein if I give an index, it returns the image in the $ImageFolder$ for which I need to compute the vanishing point. If you want to test it, please enure you give the right index and type of image like ax or 1xxx.
3. I then used one of the given function in the code file named $APPgetLargeConnectedEdges(grayIm, minLen)$ which takes in gray scale image and returns corresponding long, straight lines parameters of the format $(nlines, [x1x2y1y2thetar])$.
4. I have K-means clustering method to cluster the lines. I have used the angle subtended with the x-axis. The output of K-means in seperated into three distinct lines named line1, line2, line3.
5. I have written a helper function called $ransac_helper$ which takes the lines one by one individually. The function does ransac. Inside the function, vanishing point is computed of 2 lines selected randomly and then we compute the perpendicular distance of other lines with the vanishing point computed. If the absolute distance is lesser than threshold, we consider that point as an inlier. We perform this method for 2000 iterations and return the ransac which had the most number of inliers.
6. The $ransac_helper$ function returns the vanishing point that had the maximum number of inliers. We compute for three sets and we end up with 3 vanishing points.
7. The result is visualized as asked.

2.1.2 Results

Results from 1 series (1_001)

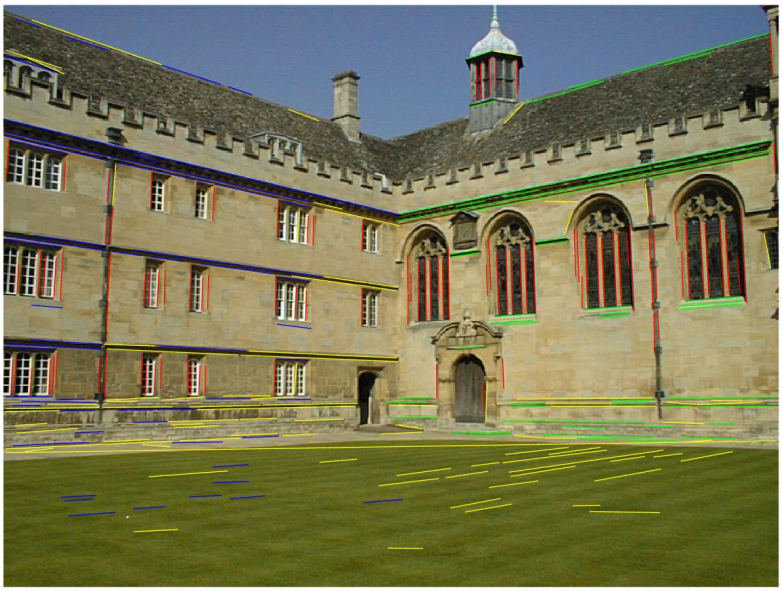

Figure 1: Color Coded Line segments

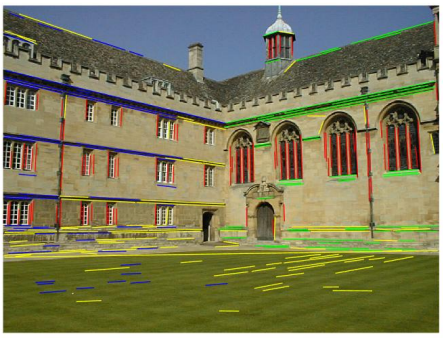

Figure 2: Left vanishing Point

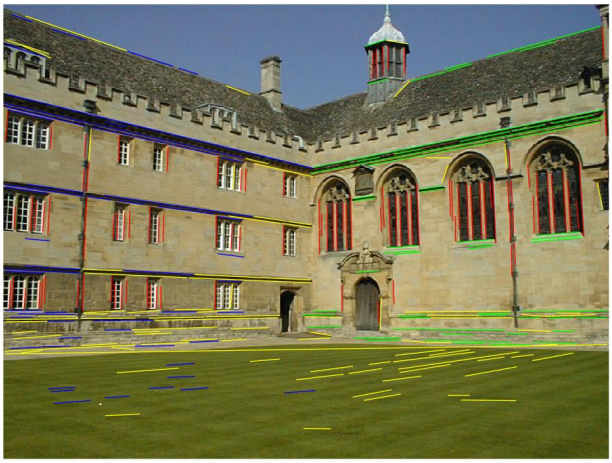

Figure 3: Right vanishing Point

Figure 4: Top vanishing Point

Figure 5: All three vanishing points together

Figure 6: Color Coded Line segments

Figure 7: Left vanishing Point

Figure 8: Right vanishing Point

Figure 9: Top vanishing Point

Figure 10: All three vanishing points together

Results from P series (P1030001)

Figure 11: Color Coded Line segments

Figure 12: Left vanishing Point

Figure 13: Right vanishing Point

Figure 14: Top vanishing Point

Figure 15: All three vanishing points together

Figure 16: Color Coded Line segments

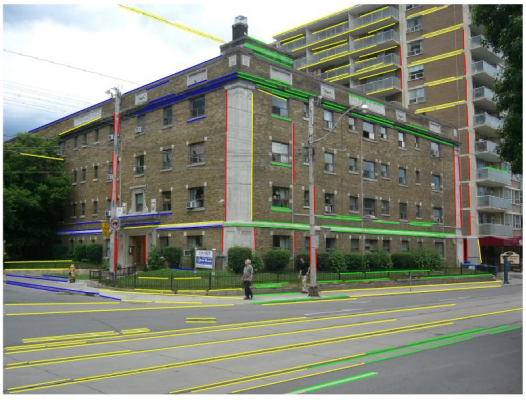

Figure 17: Left vanishing Point

Figure 18: Right vanishing Point

Figure 19: Top vanishing Point

Figure 20: All three vanishing points together

Result from Custom Image

Figure 21: Color Coded Line segments

Figure 22: Left vanishing Point

Figure 23: Right vanishing Point

Figure 24: Top vanishing Point

Figure 25: All three vanishing points together

2.2 Homographies for Plane Detection and 3D Reconstruction

2.2.1 Sparse Reconstruction

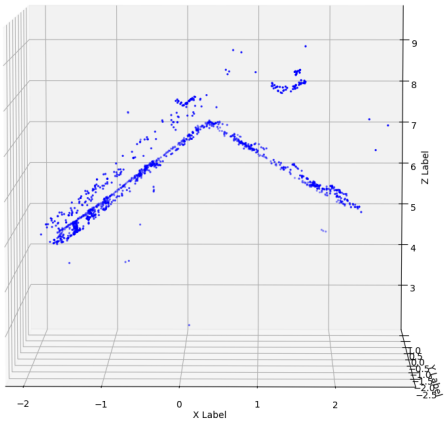

Figure 26: Sparse Point Cloud

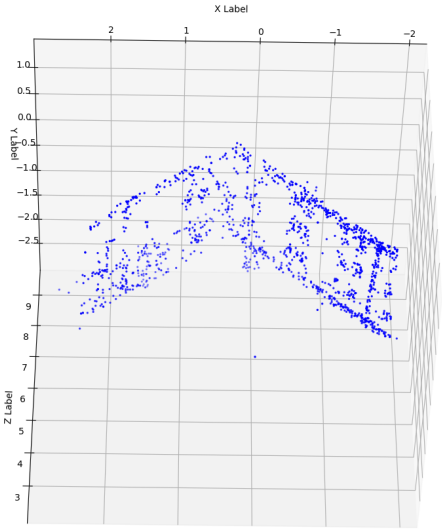

Figure 27: Sparse Point Cloud of the two major planes in 1_xxxx image series of given input

2.2.2 Dense Reconstruction

Please refer to the videos for reconstruction. Following are the results of desnse reconstruction:

Results from 1_002 and 1_003

Figure 28: Dense reconstruction of 1 series

Figure 29: Angle Between two planes for one series

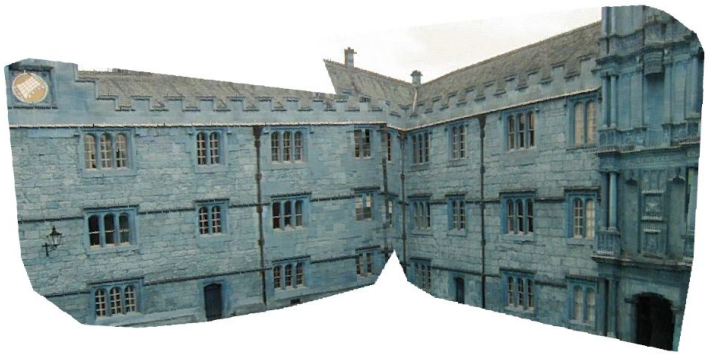

Figure 30: Dense reconstruction of 3 series

Figure 31: Angle Between two planes for three series

Figure 32: Dense reconstruction of 2 series

Figure 33: Angle Between two planes for two series

Reconstruction for custom Images: Note that the input images are labelled as numeric 5 and 6 in the submission folder.

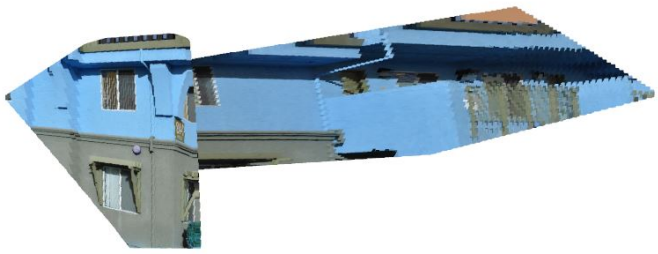

Figure 34: Dense reconstruction of custom images

Figure 35: Angle Between two planes for custom series

2.2.3 Description of Implementation

Following is the algorithm used for Plane Detection and reconstruction:

1. The first step is to take two distinct images of the same type as input. Please refer to the code main.py file.

2. Extract the camera matrices $k1$ and $k2$ that was computed in 2.1 question.

3. The next step is to do feature matching between the two images referred to as 'img1'and 'img2' respectively. I have used a sift based detector. The feature matching step returns the key-points $kp1$ and $kp2$ in img1 and img2 along with the indices. We use the keypoints to find the corresponding points in img1 and img2. These are the x and y coordinates and we append 1 to them.

4. For calculating these things - I used BRIEF.py script (also note that you would have to change the paths of the image inside main BRIEF.py script). Here I am using key point detector and descriptor algorithm to get locs1, locs2, dec1, desc2. For doing this I used brief detector. First step in doing so is to calculate DoG (Differential of gaussian) on image. Then I pass the output to computeBrief function. Here I append respective locs and docs in the list.

5. We now use ransac to find the inliers to estimate the planar homography. The function $ransacH$ which has been used from 16-720B class is used which returns the homography and inliers. The inliers will be used to the get the planes lying on the plane. We repeat the same step to get inliers and outliers in the other plane. We save the inliers and individual homography ($bestH1$ and $bestH2$ for planes 1 and 2 respectively) and use the inliers to extract the points lying on plane1 and plane2 repectively.

6. We compute convex hull using the inlier points of plane1 and plane2. Convex hull returns the extremeties of the plane in 2D which would be used for dense construction. For this, I have used $Delaunay$ to compute all the points lying inside the convex hull. We extract all the points lying within the convex hull for plane1 and plane2, and save them.

7. I am computing the Fundamental Matrix F using the function $ransacF$ function which would be used to compute the Essential Matix E.

8. We assume that initial the R or the Rotation matrix is Identity and we compute $M1$ accordingly. We then compute $M2$ using $findCorrectM$ function. Note that the function returns 4 values of $M2$ and we use the one that gives the least projection error.

9. We then triangulate using the camera matrix $C1$, $C2$, inliers in img1 and img2 to reconstruct the surface. I have used this for both sparse and reconstruction. For Dense, I directly use the Homography between the two planes to project the points of plane 1 to plane 2.