AVG Practical Sessions Report

Advanced Visual Geometry - Ecole Centrale Nantes

Justin Lee
Student ID: 200406B
justin.lee@eleves.ec-nantes.fr

Steven Palma Morera
Student ID: 200443K
steven.palma-morera@eleves.ec-nantes.fr

***Index Terms*—Projective geometry, homography, fundamental matrix, two views geometry, disparity map, visual odometry**

I. Introduction

In the present report, the analysis of the results obtained for all the three labs can be found. Each analysis contains a small explanation of the steps followed -including code snippets- and a brief discussing of the results. All the results shown in this document are of own elaboration and were obtained by running the lab scripts developed using Python 3.8.5, OpenCv 4.4.0, Numpy 1.19.4 and Matplotlib 3.3.2 on December 28th, 2020. The full repository of the lab can be found here.

II. Lab 1: Projective Geometry

In the first lab, the task was to apply an affine rectification and a metric rectification to an image, shown in Fig. 1.

Fig. 1. Initial image for Lab 1

A. Affine Rectification

Affine rectification is a transformation of a projective geometry that preserves parallelism. To find this transformation, we find the line at infinity $\mathbf{l}_\infty$ by doing the cross product of the ideal points. The ideal points are found by doing the cross product of parallel lines, which were found by doing the cross product of the points chosen on the image. Then, following Eq.1 we use the line at infinity $\mathbf{l}_\infty$ to build the homography matrix.

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_{\infty 1} & l_{\infty 2} & l_{\infty 3} \end{bmatrix} \tag{1}$$

```
hor_0 = np.cross(pts_homo[0],pts_homo[1])

pt_ideal_0 = np.cross(hor_0, hor_1)

l_inf = np.cross(pt_ideal_0,pt_ideal_1)

H = np.array([[1,0,0],[0,1,0],[l_inf[0],l_inf[1],
    l_inf[2]]])

affine_img = cv2.warpPerspective(img, H_E @ H, (img.
    shape[1], img.shape[0]))
```

Listing 1. Affine Rectification minimal example code

In Fig.2, it can be noticed how the lines formed by the chosen points are now parallel, in contrast to Fig.1 in which they aren't. Fig.2 is then now in the affine space, and therefore the parallelism was recovered.

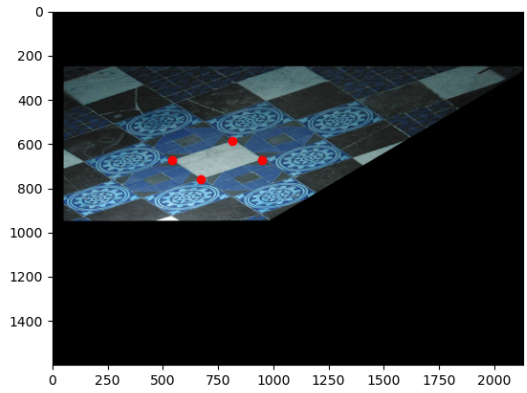

Fig. 2. Affine rectification of initial image

B. Metric Rectification

A metric rectification preserves orthogonality. This can be done using a pair of orthogonal lines. In this case, we used two parallel lines from the central white square in Fig. 1, and the two diagonal lines of the same square. Circular points and the dual conic C^*_∞ are fixed in projective transformations when they are similarity transformations.

1) Dual Conic Identification: To find the dual conic, we use the two pairs of orthogonal lines to create two constraints in this form:

$$(l'_x m'_x, l'_x m'_y + l'_y m'_x, l'_y m'_y)s = 0 \tag{2}$$

where l and m are orthogonal lines in homogeneous coordinates, and s is the null vector. To find s, we stack the two constraints into one matrix as shown in Eq. 3

$$C = \begin{bmatrix} l'_{1x}m'_{1x} & l'_{1x}m'_{1y} + l'_{1y}m'_{1x} & l'_{1y}m'_{1y} \\ l'_{2x}m'_{2x} & l'_{2x}m'_{2y} + l'_{2y}m'_{2x} & l'_{2y}m'_{2y} \end{bmatrix} \tag{3}$$

From this matrix, SVD decomposition is done to get the following matrices:

$$C = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{4}$$

The null vector s can be found by taking the last column of the transpose of $\mathbf{V}^*$:

$$s = \mathbf{V}^{*T} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{5}$$

From the null vector s, we can construct the matrix S like so:

$$\mathbf{S} = \begin{bmatrix} s_1 & s_2 \\ s_2 & s_3 \end{bmatrix} \tag{6}$$

Now, the dual conic can be constructed:

$$C'^*_\infty = \begin{bmatrix} \mathbf{S} & \mathbf{0}_{2\times1} \\ \mathbf{0}_{1\times2} & 0 \end{bmatrix} \tag{7}$$

2) Metric Rectification: Now we must compute the transformation U that moves C'^*_∞ to its canonical position, as described below:

$$C'^*_\infty = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} U^T \tag{8}$$

Note that U must be an affinity, which as explained above is a transformation that preserves parallelism. It has the following form in Equation 9.

$$U = \begin{bmatrix} \mathbf{K}_{2\times2} & \mathbf{t}_{2\times1} \\ \mathbf{0}_{1\times2} & 1 \end{bmatrix} \tag{9}$$

By substituting Eq. 9 and Eq. 7 into Eq. 8, we get Eq. 10.

$$C'^*_\infty = \begin{bmatrix} \mathbf{K}\mathbf{K}^T & \mathbf{0}_{2\times1} \\ \mathbf{0}_{1\times2} & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{S} & \mathbf{0}_{2\times1} \\ \mathbf{0}_{1\times2} & 0 \end{bmatrix} \tag{10}$$

From Eq. 10, we see that $\mathbf{S} = \mathbf{K}\mathbf{K}^T$. We can find $\mathbf{K}$ by performing eigen decomposition on $\mathbf{S}$ as shown in Eq. 11.

$$\begin{aligned} \mathbf{S} &= \mathbf{Q}\mathbf{\Sigma}\mathbf{Q}^T = \mathbf{Q}\sqrt{\mathbf{\Sigma}}(\mathbf{Q}\sqrt{\mathbf{\Sigma}})^T \\ \mathbf{K} &= \mathbf{Q}\sqrt{\mathbf{\Sigma}} \end{aligned} \tag{11}$$

Here, $\mathbf{Q}$ is a matrix where each vector is an eigen vector of $\mathbf{S}$ and $\mathbf{\Sigma}$ is a diagonal matrix which is made up of the eigenvalues of $\mathbf{S}$.

Now that we have the transformation matrix $\mathbf{U}$, we apply the homography shown in Eq.12 to the result of Section II-A and produce Fig. 3. Notice how orthogonality is now recovered.

$$H = \begin{bmatrix} \sqrt{\Sigma}^{-1}Q^T & 0_{2x1} \\ 0_{1x2} & 0 \end{bmatrix} \tag{12}$$

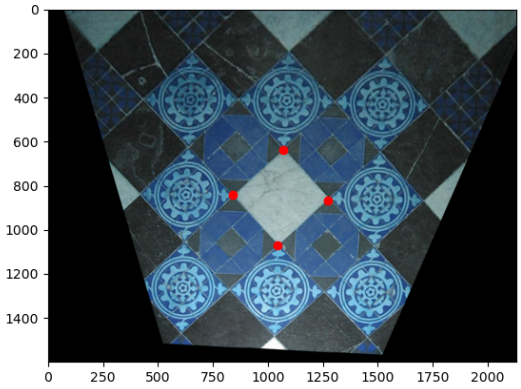

Fig. 3. Metric rectification of initial image

III. Lab 2: Two-Views Geometry

A. Computation of Epipole

The objective of this section is to compute the epipole of the image from Fig.6 and to compare it to the ground truth. In order to compute the ground truth epipole, we apply the SVD decomposition to the fundamental matrix given in the document called "chapel.00.01.F" and we keep the last column of the V matrix. The result of this epipole is shown in Eq.13.

$$epipole_{real} = \begin{bmatrix} -3234.455 \\ 266.098 \\ 1 \end{bmatrix} \tag{13}$$

On the other hand, to compute the epipole from the given image, three main steps are followed.

1) Choose manually two corresponding points in the image.
2) Compute the epipolar line from the fundamental matrix and the points chosen.
3) Compute the epipole by doing the cross product of the two epipolar lines.

```
# Compute the epipolar lines by l = F x
l= F @ x.T

# Compute the epipole from the epipolar lines
    computed
ec= np.cross(l[:,0],l[:,1])
ec /= ec[-1]
```

Listing 2. Computation of the epipolar line and epipole

In Fig.3 the epipolar lines obtained are shown. And the epipole obtained is the one shown in Eq.14.

$$epipole_{computed} = \begin{bmatrix} -3234.455 \\ 266.098 \\ 1 \end{bmatrix} \tag{14}$$

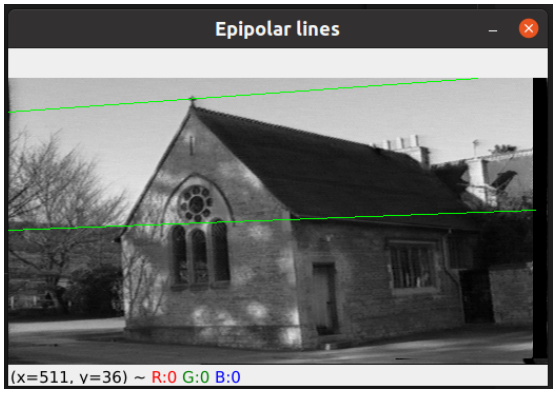

Fig. 4. Epipolar lines obtained

B. Fundamental Matrix

The purpose of this section of the lab is to estimate the fundamental matrix using the Eight Points Algorithm. First of all the fundamental matrix is a homogenous matrix of rank 2 with 7 DOF which maps pairs of image pixels from one image to another (Eq. 15). We need at least 8 pairs of points to estimate F. The two images that will be used to estimate the fundamental matrix are Figures 5 and 6. There are several steps we will use to estimate the fundamental matrix.

$$F - \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & 33 \end{bmatrix} \tag{15}$$

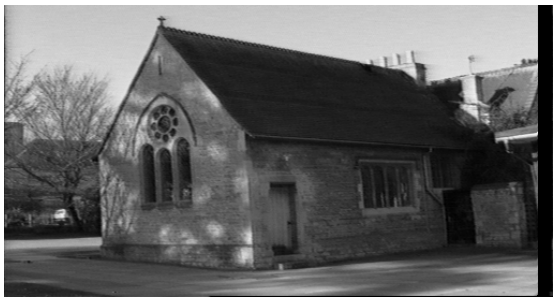

Fig. 5. Chapel image 1

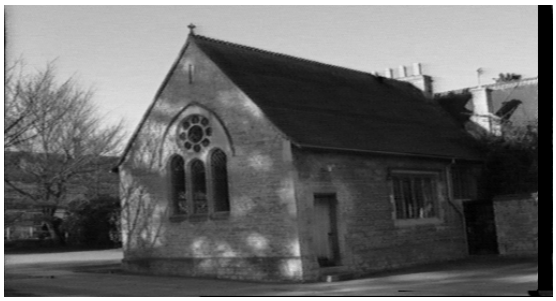

Fig. 6. Chapel image 2

1) Data Acquisition: First of all, we use an ORB descriptor to extract features from each image. We then match the features together to create pairs of points using opencv functions. We then organize the pairs into matrices, with one point in one matrix and the other point in another matrix.

2) 8-Point Algorithm: Before we can use the algorithm, we must normalize the points according to Eq. 16.

$$\begin{aligned} \hat{\mathbf{x}}_i &= \mathbf{T}\mathbf{x}_i \\ \hat{\mathbf{x}}'_i &= \mathbf{T}'\mathbf{x}'_i \end{aligned} \tag{16}$$

First, we find the center the set of points by calculating the mean of x and y $(\bar{x}, \bar{y})$. We then find the distance of every point from this center. Using the mean of these distances $\bar{b}$, we can find the scale factor of the similarity transformation in Eq. 17.

$$s = \frac{\sqrt{2}}{\bar{b}} \tag{17}$$

Now we can construct T in Eq. 18. Note that T and T' are constructed in the same way, just with different sets of points.

$$\mathbf{T} = \begin{bmatrix} s & 0 & -s\bar{x} \\ 0 & s & -s\bar{y} \\ 0 & 0 & 1 \end{bmatrix} \tag{18}$$

The normalized points are calculated by putting Eq. 18 into Eq. 16.

Now that we have the normalized points, we can construct homogeneous linear equation to estimate F:

$$\begin{aligned} A &= \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & n'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \\ \mathbf{f} &= \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} \\ A\mathbf{f} &= 0 \end{aligned} \tag{19}$$

To solve this equation, we will use SVD decomposition to factor A, and then take the last column of V^*, as $\mathbf{f}$ is the null vector of A. From f, we can get $\hat{F}$, which is an estimation of F. However, $\hat{F}$ has rank 3, while F must have rank 2. To solve this, we do SVD decomposition on $\hat{F}$, set the smallest singular value of $\hat{F}$ to zero, and reconstruct the matrix to get $\hat{F}'$. Now we denormalize the matrix in Eq. 20.

$$F = T'^T \hat{F}' T \tag{20}$$

The error in our estimated Fundamental Matrix the ground truth Fundamental Matrix is shown in Fig. 7.

```
[[-6.39013097e-07  2.34154703e-05 -5.85286781e-03]
 [-2.37760658e-05 -1.30534571e-06 -7.57681062e-04]
 [ 6.30512589e-03  7.51155354e-04  9.97961142e-02]]
```

Fig. 7. Error in F

The Fundamental matrix of the images were also computed using the OpenCV function as shown in Listing 3. The error obtained is shown in Eq.21.

```
Fransac, mask= cv2.findFundamentalMat(query_kpts,
    train_kpts, cv2.FM_RANSAC)
```

Listing 3. Funtamental matrix estimation using the OpenCV function

$$\begin{bmatrix} -6.36053768e^{-06} & -7.08622451e^{-04} & 5.74878730e^{-02} \\ 6.61082926e^{-04} & 6.64871464e^{-05} & -3.40675322e^{-01} \\ -5.51818153e^{-02} & 3.52746593e^{-01} & 1.00228938e^{00} \end{bmatrix} \tag{21}$$

C. Disparity Map and 3D Reconstruction

For this part of the lab, we used two rectified images from the KITTI dataset, shown in Figures 8 and 9.

Fig. 8. KITTI Left Image

Fig. 9. KITTI Right Image

The projection matrices and the calibration matrix are shown in Eqs. 22 and 23 respectively.

$$\begin{aligned} P_{left} &= \begin{bmatrix} 721.5377 & 0 & 609.5593 & 0 \\ 0 & 721.5377 & 172.8540 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ P_{right} &= \begin{bmatrix} 721.5377 & 0 & 609.5593 & -387.5744 \\ 0 & 721.5377 & 172.8540 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{22}$$

$$K_{left} = K_{right} = \begin{bmatrix} 721.5377 & 0 & 609.5593 \\ 0 & 721.5377 & 172.8540 \\ 0 & 0 & 0 \end{bmatrix} \tag{23}$$

We perform dense stereo matching using OpenCV functions. From this we get a disparity map, and we want to find the depth of the pixels whose disparity is positive. The depth of a pixel is shown in Eq. 24, where Bf is the base line $\times$ focal length, and d is the disparity.

$$Z = \frac{Bf}{d} \tag{24}$$

We now want to normalize the coordinates of every pixel whose disparity is positive. To do this we use Eq. 25.

$$p_{normalized} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{25}$$

To calculate the 3D coordinates we use Eq.

$$p_{3D} = Z * p_{normalized} \tag{26}$$

We then get the colour of every pixel whose disparity is positive, and we display the point cloud in Fig. 10.

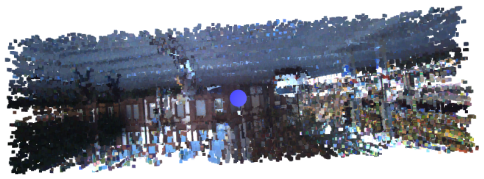

Fig. 10. 3D Reconstruction of the Image

IV. Lab 3: Homography-Based Visual Odometry

In this lab, we were asked to determine the pose of the camera for a given camera frame C^i. The world frame is chosen as the first camera frame C^0. We want to compute a homography based off two images of a plane. It will map a pixel x^0 from the first frame to the x^i in the i-th frame as in Eq. 27

$$x^i = \mathbf{H} x^0 \tag{27}$$

The plane of the first camera frame C^0 can be expressed as:

$$\mathbf{X}^T \mathbf{n} = d \tag{28}$$

The homography can then be computed using the following formula:

$$\mathbf{H} = \mathbf{R} + \mathbf{t} \frac{\mathbf{n}^T}{d} \tag{29}$$

To find the pose of the camera, it requires three steps:

1) Find the matches of keypoints on two images of the plane
2) Determine $\mathbf{R}, \mathbf{t}, \mathbf{n}$ from the homography $\mathbf{H}$
3) Prune the wrong candidates

A. Matching Keypoints

To match the keypoints we find ORB features in the source frame and the current frame and find matches between them.

```
            self.incoming_kpts, self.incoming_desc =
     self.orb.detectAndCompute(gray,None)  # TODO
            matches = self.matcher.match(self.
    src_desc,self.incoming_desc)  # TODO
```

Listing 4. Matching keypoints code

B. Decomposition of $\boldsymbol{H}$

The homography $\mathbf{H}$ is found using an OpenCV function:

```
        mat_H, mask = cv2.findHomography(src_pts,
    incoming_pts, cv2.RANSAC, self.
    homography_ransac_threshold) # TODO
```

Listing 5. computation of Homography from point correspondences

We now find the inliers on the source image with respect to $\mathbf{H}$ and normalize these points. $\mathbf{H}$ is now decomposed into candidates for $\mathbf{R}, \mathbf{t}, \mathbf{n}$ using another OpenCV function:

```
        num_candidates, rots, trans, normals = cv2.
    decomposeHomographyMat(mat_H, self.
    camera_intrinsic)  # TODO
```

Listing 6. Decomputation of H

C. Pruning of Wrong Candidates

To prune the wrong candidates, we want to get rid of the ones that have a negative depth for minimum 1 inlier. To do this, we can find the sign of the ratio between the d and Z with the following equation:

$$\left(\mathbf{K}^{-1}\begin{bmatrix}u\\v\\1\end{bmatrix}\right)^{T}\begin{bmatrix}n_x\\n_y\\n_z\end{bmatrix} = \frac{d}{Z} \tag{30}$$

This is done with the code below:

```
                d_over_z = src_pts_normalized @ n #
    TODO: result should be an array of <float:
    num_inliers>
                # check if any ratio is negative
                is_valid = np.all(d_over_z > 0)  #
    TODO: True if every inlier has positive depth,
    False otherwise
```

Listing 7. Pruning candidates with negative depth

There may be two candidates left after this is done. To remove the final one, we keep the candidate with $\mathbf{n}$ which is closer to the current estimation of the plane's normal vector.

```
                angle_0 = np.arccos(np.dot(normals
    [0].T,self.plane_normal_src)) # TODO: compute
    angle between 1st candidate of normal vector
    with self.plane_normal_src
                angle_1 = np.arccos(np.dot(normals
    [1].T,self.plane_normal_src)) # TODO: compute
    angle between 2nd candidate of normal vector
    with self.plane_normal_src
                del_idx = 0 if angle_0 > angle_1
    else 1
```

Listing 8. Keeping the optimal candidate

Now that we have an $\mathbf{R}$ and $\mathbf{t}$, we must scale the $\mathbf{t}$, as the translation vector from $\mathbf{H}$ is actually $\mathbf{t}/d$. All we need to multiply $\mathbf{t}$ by d to scale it up. To find d, we can use the following equation:

$$d = \mathbf{n}^T {}^{C^0}\mathbf{t}_p \tag{31}$$

D. Updating Source Frame

Note that we also have the option of updating the source frame with the current frame. To do this, we set the ORB feature points of the current frame to the source frame. We also must recalculate the plane's distance with the new frame. First of all, we calculate the depth by finding the d/Z as in Eq. 30 and multiply the inverse of this by d. Now we find the 3D coordinates of the source points by multiplying the normalized points by the depth. To map these source points to the current frame, we do the following:

$$\mathbf{p}_{current} = \mathbf{R} \cdot \mathbf{p}_{src} + \mathbf{t} \tag{32}$$

To find the new d, we take the mean projection of every point onto the $\mathbf{n}$:

$$d = mean(\mathbf{p}_{current} \cdot \mathbf{n}) \tag{33}$$

E. Results

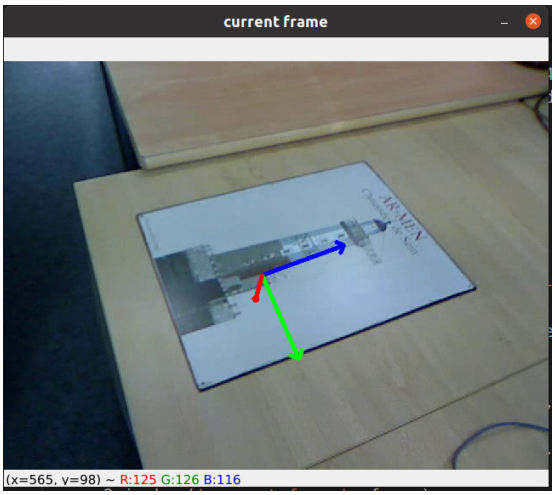

Fig. 11. Example frame of the Lab3 result video

The Fig.11 shows a single frame of the resultant video. The axis drawn remains mostly immobile even if the camera is moving. This is only possible if the rotation matrix and translation vector between each frame is correctly computed such that the axis drawn is adjusted accordingly. The video obtained is very similar to the one found here.

REFERENCES

[1] Dao, Minh-Quan. (2020). AVG Practical Session 1: Projective Geometry
[2] Dao, Minh-Quan. (2020). AVG Practical Session 2 : Two-views Geometry
[3] Dao, Minh-Quan. (2020). AVG Practical Session 3: Homography-based visual odometry.