COMPUTER VISION

REPORT 2

BRICE Chloe - 9IE24001M

CONTENT

Project structure : What are the steps and how the code is implemented

Epipolar Geometry : Results

Photometric : Normals computation

PROJECT STRUCTURE

This project is implemented both in C++ and Python, and uses OpenCV library. We first run the main.cpp file which computes the essential matrix with the 5-points algorithm with OpenCV. The essential matrix is stored in a txt file. Then we run the main.py which uses the essential matrix to compute and enlighten the epipolar geometry of those stereo images. We will be able to compare the 8-points and the 5-points algorithm.

The 2nd part of the python code aims to compute the normals of an object's surface. We use the photometric method, detailed below.

5-PTS ALGORITHM

We use the 5-points algorithm : the points' coordinates are described in a .txt file, which is parsed by the program. Then we can begin the computation.

To do so, we use the OpenCV findEssentialMatrix function which uses RANSAC algorithm for outlier detection and exclusion. This function can take more than 5 points to do the computation. Otherwise, it doesn't return a single 3x3 matrix but for instance, with 5 points, it returns 4 essential matrices. After research, it appears that it can be the case when there is an ambiguity when computing E with the points, so it offers multiple E matrices solution. In our case, we always get multiple E matrices from this OpenCV function ; so we choose only one to compute the epipolar lines. When we compare the results with different E, they are always similar. So the choice of the E matrix computed seems not important for the next questions.

```
E = cv::findEssentialMat(points1_cv,points2_cv,K1,cv::RANSAC, 0.999, 1, mask);
```

Intrinsic matrix

RANSAC level of confidence

Thresold

Example of E matrix computed :

[0.0001171500034283975, 0.0017266482602639 06, -0.6553229773048173;
-0.001065231688666 76, -0.0009928674705603682, -0.2656128842185946;
0.3884156777129563, 0.5908713803099762, 0.0008653600956500691]

5-PTS ALGORITHM

NB : the algorithm can also work with 8 points and in this case, only return one possible essential matrix:

Number of points for the Algorithm : 8
Points loaded : (img1.x ; img1.y / img2.x ; img2.y)
1040 ; 566 / 979 ; 606
632 ; 271 / 733 ; 267
423 ; 406 / 457 ; 400
303 ; 313 / 369 ; 304
911 ; 157 / 862 ; 149
462 ; 546 / 476 ; 540
20 ; 331 / 111 ; 316
956 ; 379 / 888 ; 395

Essential matrix E :
[1.091186508166207e-06, 4.777227155531705e-05, -0.01571011459474853;
 0.00011781932713447 64, -0.0001040222162930705, -0.7069322560696821;
 -0.02155074352172198, 0.7067782571346221, -0.00010647845065354 44]
Dimensions : [3 ; 3]

There are 1 possible essential matrix :
Essential matrix E1 :
[1.091186508166207e-06, 4.777227155531705e-05, -0.01571011459474853;
 0.00011781932713447 64, -0.0001040222162930705, -0.7069322560696821;
 -0.02155074352172198, 0.7067782571346221, -0.00010647845065354 44]

EPIPOLAR GEOMETRY

STEREO IMAGES

The 5 points used for computation are in color, the others are in cyan.

EPIPOLAR LINES

Firstly, we can compute F with the relation. $\mathbf{E} = \mathbf{K'}^{T}\mathbf{F}\mathbf{K}$

A-2 : Compute E # ##

Number of points loaded : 8
[[1040, 566], [632, 271], [423, 406], [303, 313], [911, 157], [462, 546], [20, 331], [956, 379]]
[[979, 606], [733, 267], [457, 400], [369, 304], [862, 149], [476, 540], [111, 316], [888, 395]]

F of report 1 = [[3.23484509e-07 -8.24416313e-06 3.16293391e-03]
[2.18257454e-05 -2.78169850e-06 4.09953385e-02]
[-7.12409269e-03 -4.52513760e-02 9.98103671e-01]]

Give the name of the file with the matrix E : matrix_E1.txt

E loaded = [[1.17150003e-04 1.72664826e-03 -6.55322977e-01]
[-1.06523169e-03 -9.92867471e-04 -2.65612884e-01]
[3.88415678e-01 5.90871380e-01 8.65360096e-04]]

New F = [[1.17150003e-04 1.72664826e-03 -6.55322977e-01]
[-1.06523169e-03 -9.92867471e-04 -2.65612884e-01]
[3.88415678e-01 5.90871380e-01 8.65360096e-04]]

EPIPOLAR LINES

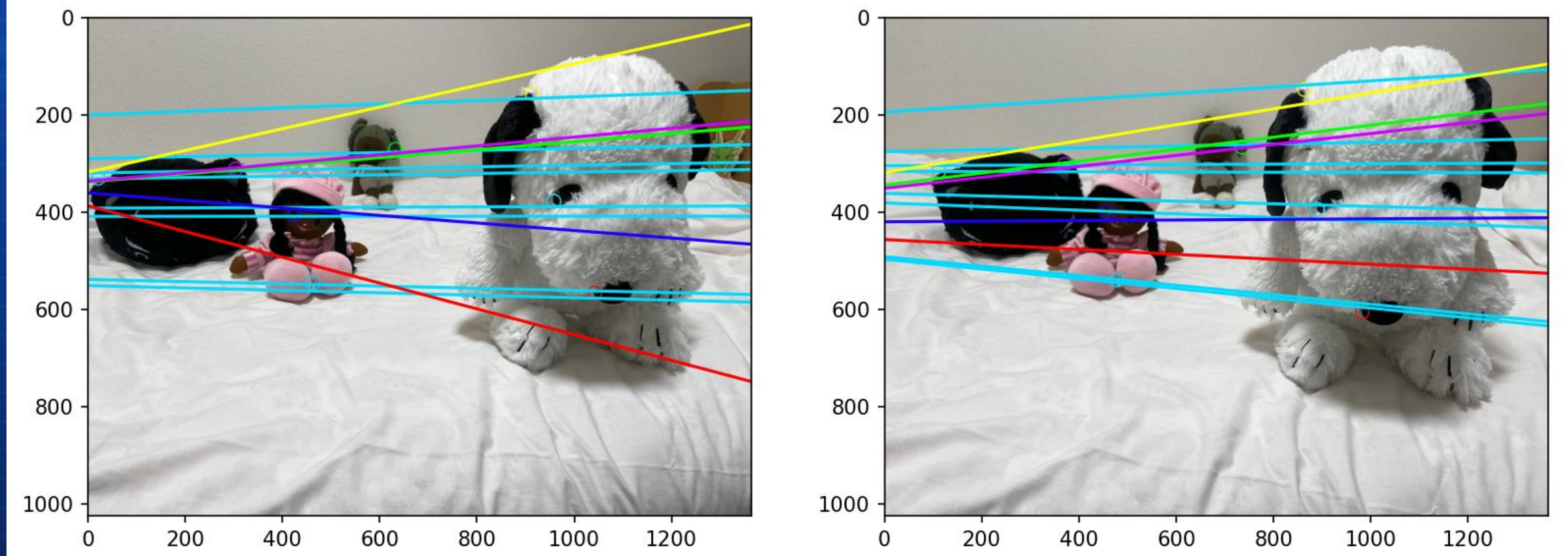
Then we use F to compute the epipolar lines, likewise in the report 1.

```
### ## # A-2 : Compute E # ## ###


Number of points loaded :  8
[[1040, 566], [632, 271], [423, 406], [303, 313], [911, 157], [462, 546], [20,
331], [956, 379]]
[[979, 606], [733, 267], [457, 400], [369, 304], [862, 149], [476, 540], [111,
316], [888, 395]]

F of report 1 =  [[ 3.23484509e-07 -8.24416313e-06  3.16293391e-03]
 [ 2.18257454e-05 -2.78169850e-06  4.09953385e-02]
 [-7.12409269e-03 -4.52513760e-02  9.98103671e-01]]


Give the name of the file with the matrix E : matrix_E1.txt

 E loaded =  [[ 1.17150003e-04  1.72664826e-03 -6.55322977e-01]
 [-1.06523169e-03 -9.92867471e-04 -2.65612884e-01]
 [ 3.88415678e-01  5.90871380e-01  8.65360096e-04]]

 New F =  [[ 1.17150003e-04  1.72664826e-03 -6.55322977e-01]
 [-1.06523169e-03 -9.92867471e-04 -2.65612884e-01]
 [ 3.88415678e-01  5.90871380e-01  8.65360096e-04]]
```

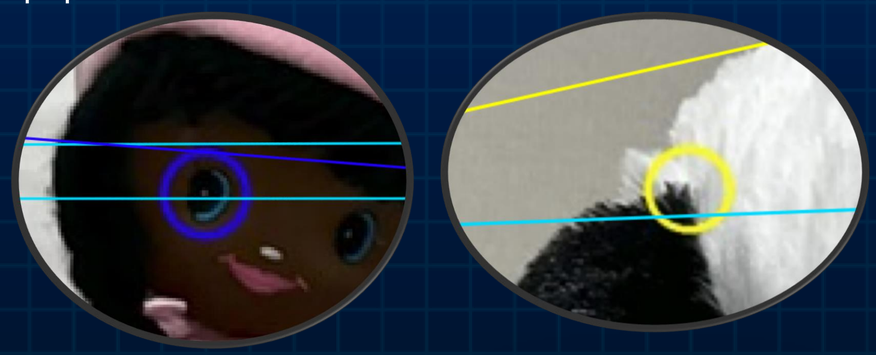
```python
# Epipolar lines compute with new_F
def draw_epipolar_line_new_F(new_F, nb_elements, points1, points2, colors, img1, img2, ax1, ax2):

    for i in range(nb_elements):

        # We obtain equations of both epipolar lines
        a1 = new_F[0][0]*points2[i][0] + new_F[0][1]*points2[i][1] + new_F[0][2]
        b1 = new_F[1][0]*points2[i][0] + new_F[1][1]*points2[i][1] + new_F[1][2]
        c1 = new_F[2][0]*points2[i][0] + new_F[2][1]*points2[i][1] + new_F[2][2]

        a2 = new_F[0][0]*points1[i][0] + new_F[1][0]*points1[i][1] + new_F[2][0]
        b2 = new_F[0][1]*points1[i][0] + new_F[1][1]*points1[i][1] + new_F[2][1]
        c2 = new_F[0][2]*points1[i][0] + new_F[1][2]*points1[i][1] + new_F[2][2]

        x1 = np.linspace(0, img1.size[0], img1.size[1])
        y1 = (-a1*x1 - c1) / b1
        ax1.plot(x1, y1, color=colors[i])

        x2 = np.linspace(0, img2.size[0], img2.size[1])
        y2 = (-a2*x2 - c2) / b2
        ax2.plot(x2, y2, color=colors[i])
```

EPIPOLAR LINES

Here are the epipolar lines represented : the lines in cyan are the ones computed with the F matrix of the report 1.



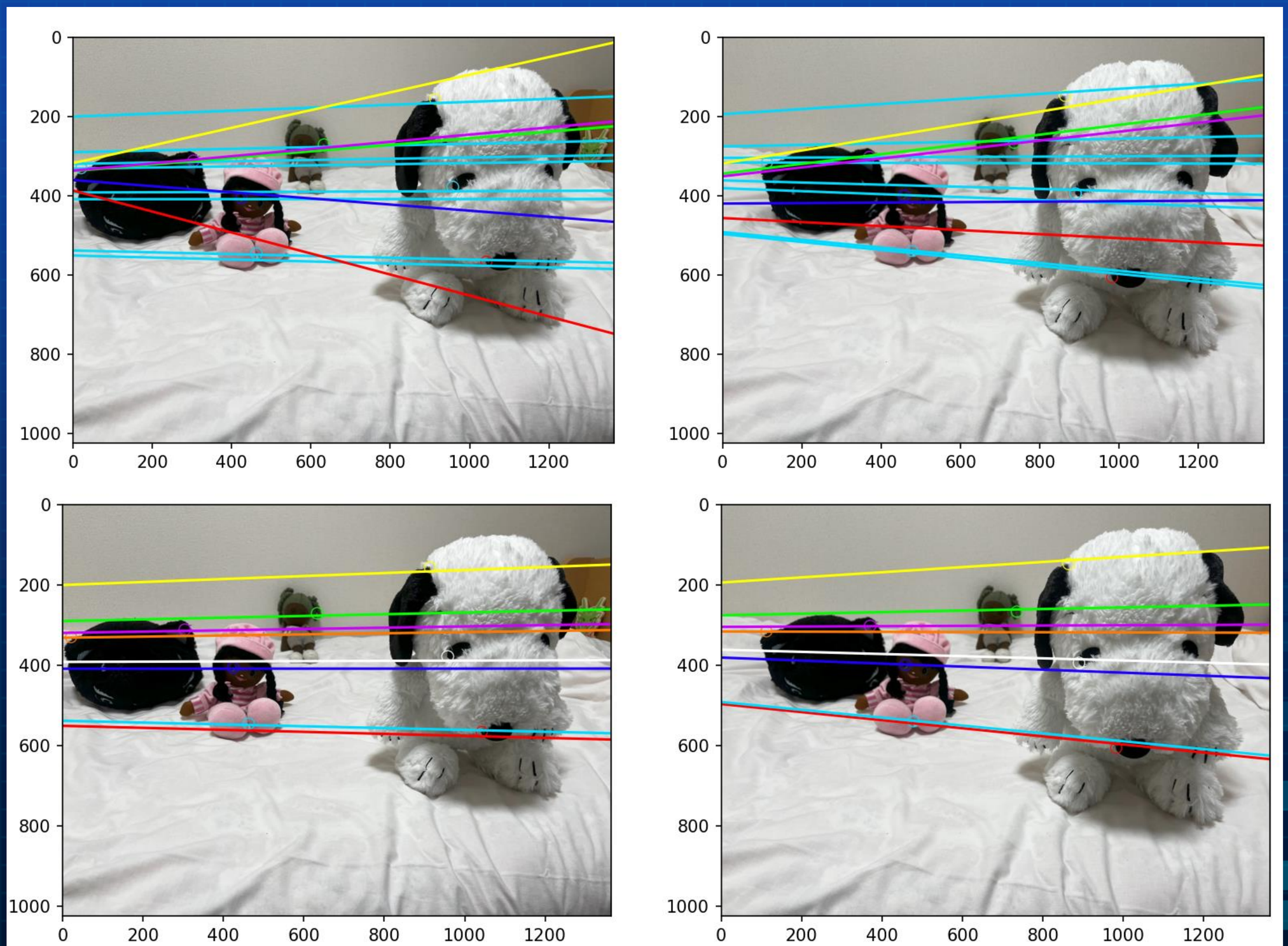
We can see that the results are less accurate with the 5-points-algorithm function of OpenCV than the 8-pts algorithm. The lines of the right image seems to converge where the image on the left was taken, but it isn't the case for the left image. Also, the epipolar lines are sometimes close to the corresponding point, but not everywhere.



EPIPOLAR LINES

For recall :

5-pts algorithm



8-pts algorithm

PHOTOMETRIC – 3D INFORMATION

LIGHT SOURCES

Here, we use images of the same object, illuminated with light sources at different locations in the space. Each image corresponds to one light source and location.

By handling the information given by each picture, we can compute the normals at the object's surface and recover its 3D surface.

Three lights are necessary to do this computation, and we can even use more to have better results.

Here we focus on three light sources.

Example of light sources location :

0.403259 0.480808 0.778592
0.0982272 0.163712 0.981606
-0.0654826 0.180077 0.98147

NORMALS COMPUTATION

What we have to do first, is get the three illuminance values for each pixel of the three images. To do so, we convert the images to grayscale, and apply a mask to compute only the image area displaying the object.

Then, with the light sources coordinates, we can find the normal coordinates at each pixel with the equation.

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \frac{\rho}{\pi} \underbrace{\begin{bmatrix} s_{x_1} & s_{y_1} & s_{z_1} \\ s_{x_2} & s_{y_2} & s_{z_2} \\ s_{x_3} & s_{y_3} & s_{z_3} \end{bmatrix}}_{S_{3\times3} \text{ (Known)}} \mathbf{n} \quad \Rightarrow \quad \begin{cases} I = S\mathbf{N} \\ \text{where: } \mathbf{N} = \frac{\rho}{\pi}\mathbf{n} \end{cases}$$

Solution: $\mathbf{N} = (S)^{-1} I$

Surface Normal: $\mathbf{n} = \frac{\mathbf{N}}{\|\mathbf{N}\|}$ Albedo: $\frac{\rho}{\pi} = \|\mathbf{N}\|$

https://snawarhussain.com/blog/computer%20vision/python/tutorial/photometric-stereo-lambertian-model/#photometric-stereo

```python
# For each pixel
for x in range(w):
    for y in range(h):

        # And each of the 3 images
        for i in range(len(gray_imgs)):
            I[i] = gray_imgs[i][y][x]

        # Use the formula to compute N
        S_inv = np.linalg.inv(S_t)
        N = np.dot(S_inv, I).T
        N_norm = np.linalg.norm(N, axis=1)

        rho = N_norm * math.pi

        # Define a luminosity to display normals
        N_gray = N[0] * 0.30 + N[1] * 0.35 + N[2] * 0.35
        N_gray_norm = np.linalg.norm(N_gray)
        if N_gray_norm == 0:
            continue

        # Store the normals' values of the image
        img_normal[y][x] = N_gray / N_gray_norm

# Convert to rgb for colored representation
return ((img_normal * 0.5 + 0.5) * 255).astype(np.uint8)
```

Then we can display the 3D surface by defining luminosity coefficients for each of N's coordinates.

CLOSE LIGHT SOURCES

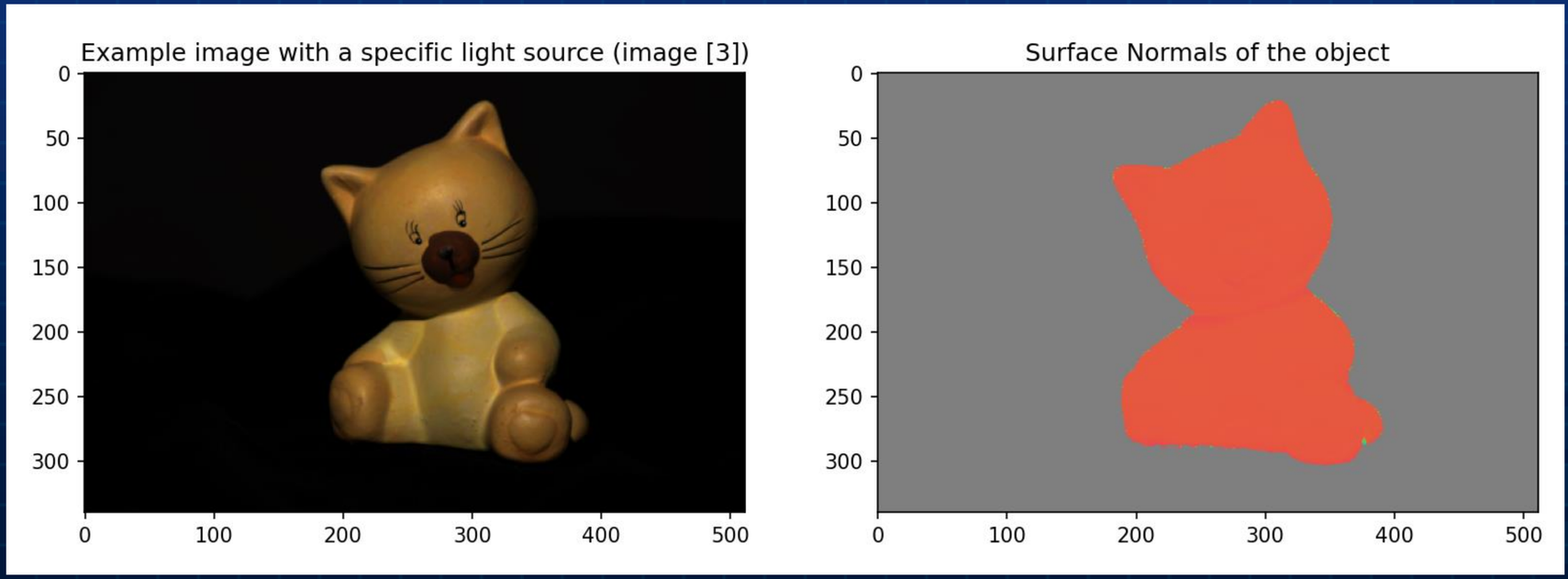
B-2 : Photometric Stereo Method # ##

Type the camera numbers (between 0 and 11, need 3 cameras)

Camera number you want to use : 3
Camera number you want to use : 5
Camera number you want to use : 11

S_list :
[[-0.127999, 0.431998, 0.892745], [-0.110339, 0.53593, 0.837021], [-0.16119, 0.354617, 0.921013]]

Here we see that with close light sources in the space, we can't get a lot of information about the object's surface ; we could have expected that, because as lights are close in the space, they only provide similar information and not complementary information on 3D surface.



DISTANT LIGHT SOURCES

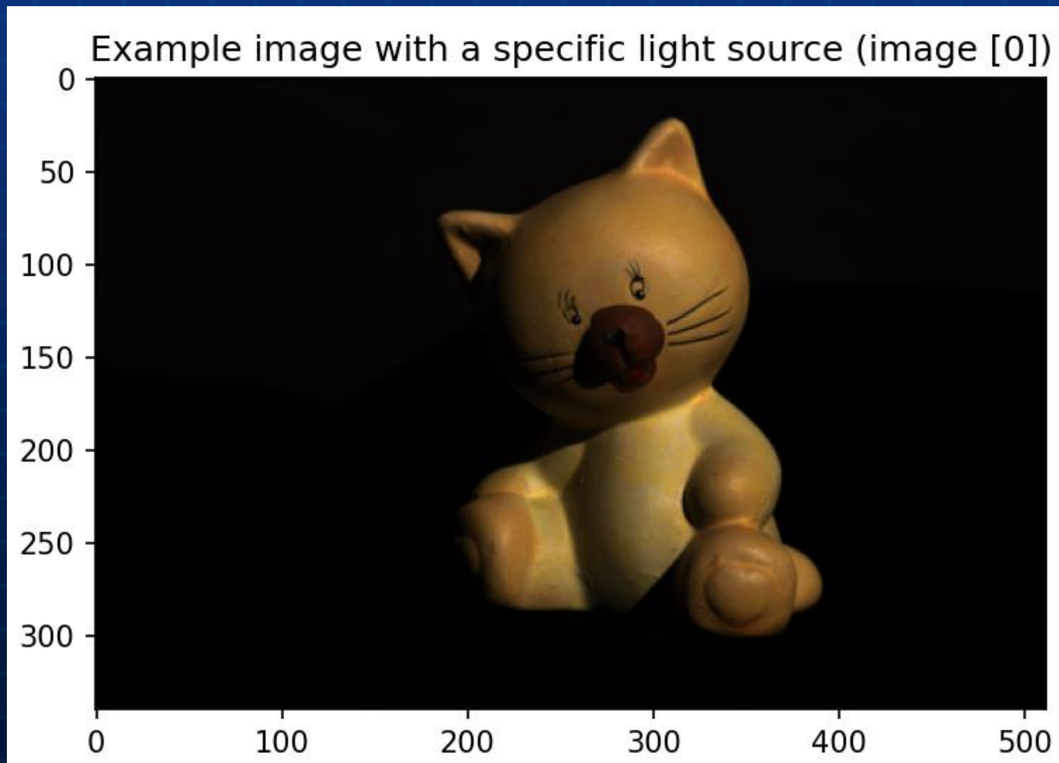
B-2 : Photometric Stereo Method # ##

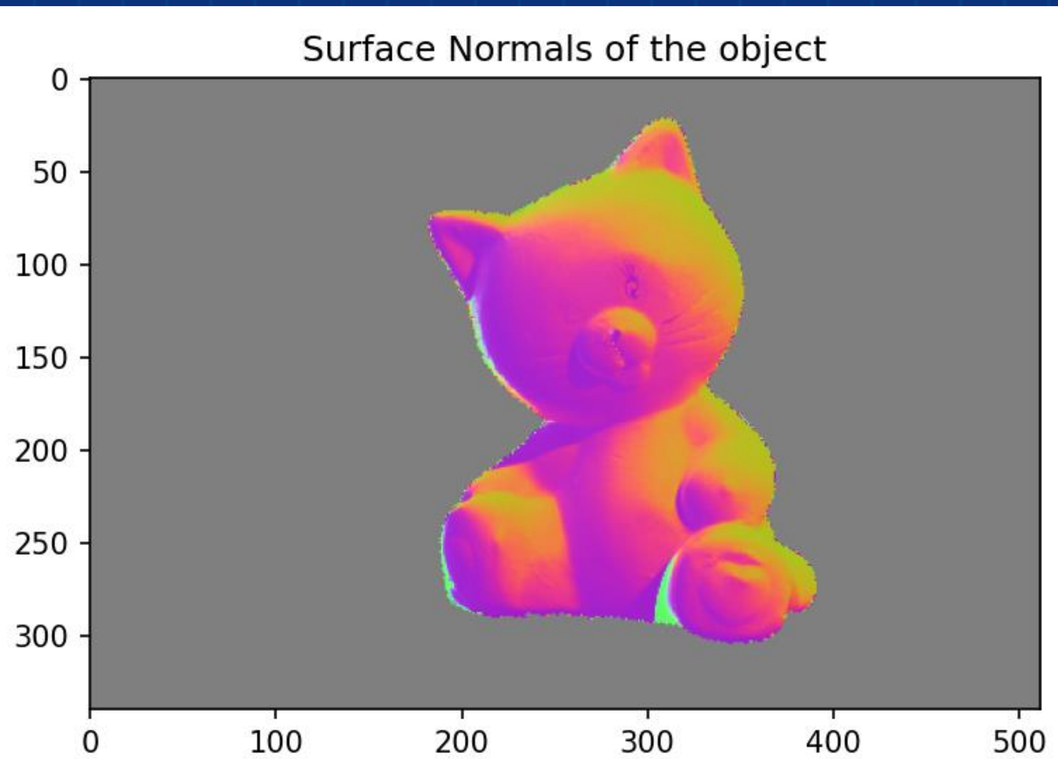
Type the camera numbers (between 0 and 11, need 3 cameras)

Camera number you want to use : 0
Camera number you want to use : 1
Camera number you want to use : 2

S_list :
[[0.403259, 0.480808, 0.778592], [0.0982272, 0.163712, 0.981606], [-0.0654826, 0.180077, 0.98147]]

With distant light sources, we can clearly recover the object's surface.





CONCLUSION

With only 5 points, we have been able to compute the essential matrix; however, when we compare our results with the 8-points algorithm, we had better results regarding epipolar geometry.

We can explain the occasional poor results with many factors:
- OpenCV algorithm for the 5-points algorithm which can sometimes be a « black box », and whose parametrization could be improved
- 8-pts algorithm is more robust to noise, because it uses more points
- The post-processing of the phone's camera
- We didn't normalize the coordinates

Then we have been able to recover a 3D surface thanks to only 3 images taken with different light sources, using the photometric method.