COMPUTER VISION REPORT ASSIGNMENT #1

Evahn LE GAL

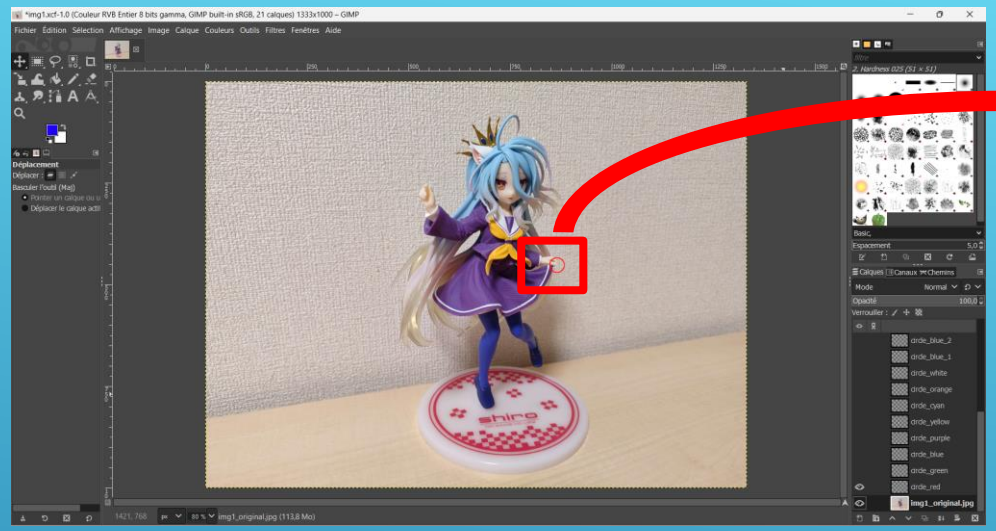
9IE24004S

05/12/2024

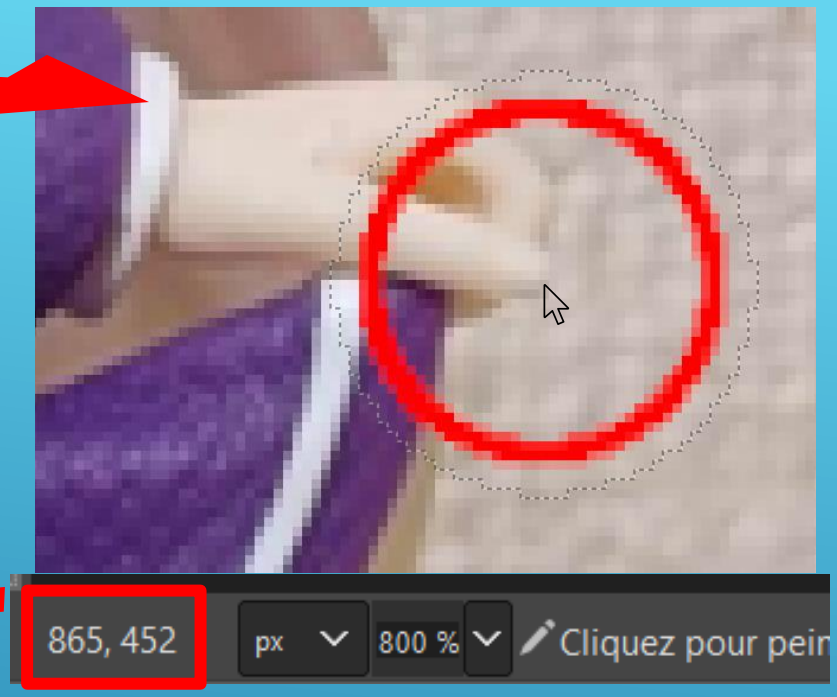
PLAN

- Structure of the Program / Sets of Data
- Epipolar Lines
- Epipolar Lines of Not Computed Points
- More Than 8 Points
- Epipoles
- Results

All the code and the images (especially those that I was not able to integrate with this report) can be find on my GitHub repository : https://github.com/Vurremt/report1



Gimp (2.10) : Image 1333 px * 1000 px



We draw the circle with the tool in Gimp

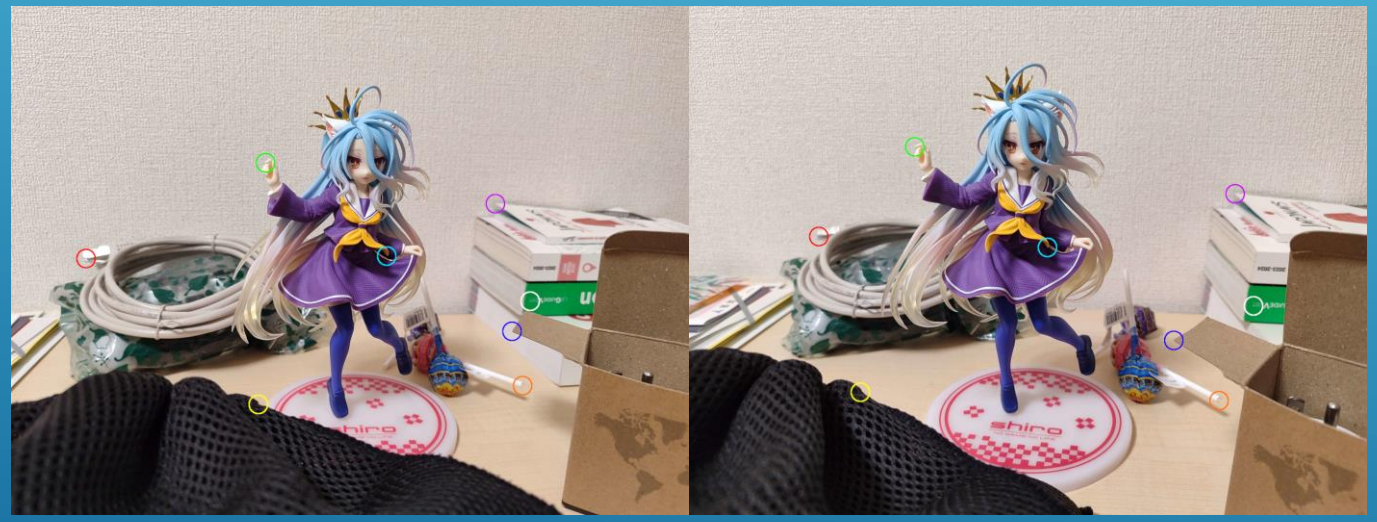
```
img1[ 865 ; 452 ]/img2[ 826 ; 457 ]/ff0000/red
img1[ 536 ; 259 ]/img2[ 474 ; 242 ]/04ff00/green
img1[ 653 ; 371 ]/img2[ 593 ; 351 ]/1f00ff/blue
img1[ 575 ; 636 ]/img2[ 545 ; 586 ]/cf00ff/purple
img1[ 711 ; 791 ]/img2[ 681 ; 766 ]/feff00/yellow
img1[ 754 ; 277 ]/img2[ 681 ; 266 ]/00daff/cyan
img1[ 650 ; 147 ]/img2[ 611 ; 138 ]/ff7900/orange
img1[ 861 ; 343 ]/img2[ 827 ; 343 ]/ffffff/white
```

We do the same for the second image and the others points

CONSTRUCT 8 POINTS ON A IMAGE

same_plane

A set of images without depth, with all 8 points on approximately the same plane



not_same_plane

A set of images with objects in the foreground and background, with the 8 points on completely different planes

2 SETS OF DIFFERENT IMAGE TYPES

Sometimes, we will use examples of “perfect” images

STRUCTURE OF THE PROGRAM

- Structure of the program: A main.py file for the different steps, and sub-files for each question
- We choose the current folder (seen previously), all other files have the same names, so we only specify the folder

```
# Name of current folder
folder_name = "./img/not_same_plane/"
```

- A program in C++ to calculate and generate a large number of points from our 2 images (A-6)

`list_of_points.txt` :

Coord. in image 1 | Coord. in image 2 | Associated colors

```
img1[ 865 ; 452 ]/img2[ 826 ; 457 ]/ff0000/red
img1[ 536 ; 259 ]/img2[ 474 ; 242 ]/04ff00/green
img1[ 653 ; 371 ]/img2[ 593 ; 351 ]/1f00ff/blue
img1[ 575 ; 636 ]/img2[ 545 ; 586 ]/cf00ff/purple
img1[ 711 ; 791 ]/img2[ 681 ; 766 ]/feff00/yellow
img1[ 754 ; 277 ]/img2[ 681 ; 266 ]/00daff/cyan
img1[ 650 ; 147 ]/img2[ 611 ; 138 ]/ff7900/orange
img1[ 861 ; 343 ]/img2[ 827 ; 343 ]/ffffff/white
```

Colors in letters for humans only

Function `load_points()`

It separates the file in 3 arrays :

- points1 : coord. of points in img1
- points2 : coord. of points in img2
- colors : color in img 1 and 2

```
Number of points loaded :  8
[[147, 496], [498, 307], [983, 639], [950, 388], [485, 782], [738, 490], [1004, 746], [1019, 580]]
[[255, 452], [443, 276], [953, 656], [1073, 368], [337, 758], [705, 474], [1042, 775], [1109, 589]]
```

`points1[i]` / `points2[i]` / `colors[i]` represent the same point " i "

DATA EXTRACTION

A-2 : COMPUTE F

- After extracting **points1** and **points2**, we can construct A such that :

$$\begin{matrix} x_1, y_1 = points1[1] \\ \vdots \\ x_1, y_1 = points1[8] \end{matrix} \begin{matrix} /x_2, y_2 = points2[1] \\ \\ /x_2, y_2 = points2[8] \end{matrix} \begin{pmatrix} x_1x_2 & x_1y_2 & x_1 & y_1x_2 & y_1y_2 & y_1 & x_2 & y_2 & 1 \\ \vdots & & & & \vdots & & & & \\ x_1x_2 & x_1y_2 & x_1 & y_1x_2 & y_1y_2 & y_1 & x_2 & y_2 & 1 \end{pmatrix} = \mathrm{A}$$

- We perform a SVD decomposition to find U, Σ and V^T:

```
U, Σ, Vt = np.linalg.svd(A)
```

- We reshape V^T into F1 and compute again a SVD decomposition :

```
U1, Σ1, V1t = np.linalg.svd(F1)
```

- We force the rank 2 of the matrix by setting the smallest eigenvalue of Σ to 0, and by constructing our F with this Σ (transformed into a diagonal matrix) :

```
Σ1[-1] = 0
Σ1_diag = np.diag(Σ1)

# Reconstruction of F
F = np.dot(U1, np.dot(Σ1_diag, V1t))
```

Note: The program can take any number of points above 8, it depends on the number of lines in the .txt (for question A-6)

Note: At each step of the previous function, the program asks us for the matrices to display

Observation: We do not use normalization for reasons of simplicity, we therefore expect approximations in the results

Results

A :

```
Print A ?
[y/n] : y
[[3.748500e+04 6.644400e+04 1.470000e+02 1.264800e+05 2.241920e+05
  4.960000e+02 2.550000e+02 4.520000e+02 1.000000e+00]
 [2.206140e+05 1.374480e+05 4.980000e+02 1.360010e+05 8.473200e+04
  3.070000e+02 4.430000e+02 2.760000e+02 1.000000e+00]
 [9.367990e+05 6.448480e+05 9.830000e+02 6.089670e+05 4.191840e+05
  6.390000e+02 9.530000e+02 6.560000e+02 1.000000e+00]
 [1.019350e+06 3.496000e+05 9.500000e+02 4.163240e+05 1.427840e+05
  3.880000e+02 1.073000e+03 3.680000e+02 1.000000e+00]
 [1.634450e+05 3.676300e+05 4.850000e+02 2.635340e+05 5.927560e+05
  7.820000e+02 3.370000e+02 7.580000e+02 1.000000e+00]
 [5.202900e+05 3.498120e+05 7.380000e+02 3.454500e+05 2.322600e+05
  4.900000e+02 7.050000e+02 4.740000e+02 1.000000e+00]
 [1.046168e+06 7.781000e+05 1.004000e+03 7.773320e+05 5.781500e+05
  7.460000e+02 1.042000e+03 7.750000e+02 1.000000e+00]
 [1.130071e+06 6.001910e+05 1.019000e+03 6.432200e+05 3.416200e+05
  5.800000e+02 1.109000e+03 5.890000e+02 1.000000e+00]]
```

U, Σ and V^T :

```
U =  [[ 0.06122602  0.2787305  -0.33551336  0.80881292 -0.16825603  0.30292417
  -0.16107505 -0.07612367]
 [ 0.10251021 -0.01372672  0.03667681 -0.03104813 -0.7226964  -0.28686802
   0.26286523 -0.55974333]
 [ 0.45532324  0.04203354  0.2690964  -0.29496375 -0.08929418  0.62675851
  -0.39405241 -0.27462784]
 [ 0.37333607 -0.50910302 -0.65389428 -0.10319325  0.0292601  -0.22499995
  -0.33243377 -0.0348479 ]
 [ 0.19521577  0.74882631 -0.46199337 -0.39737772  0.05069084 -0.06206091
   0.14636799  0.04425301]
 [ 0.25267133  0.02043416  0.13743203 -0.01250829 -0.56331142 -0.059821
  -0.10325797  0.76494125]
 [ 0.54319665  0.21622828  0.39038321  0.27391789  0.30748368 -0.5480836
  -0.16658695 -0.08975533]
 [ 0.49196479 -0.2307707  -0.03702038  0.11878     0.1616623   0.27325215
   0.76113742  0.09541866]]

Σ =  [2.97691295e+06 6.53775136e+05 1.44994868e+05 7.08742425e+04
 5.80593265e+02 8.25891336e+01 1.22133111e+01 1.20153221e+00]

V^T =  [[ 7.12018210e-01  4.43539771e-01  7.35706756e-04  4.47378882e-01
   3.10085598e-01  4.92011918e-04  7.56171985e-04  4.93978545e-04
   8.31547352e-07]
 [-5.71618340e-01  2.72166876e-01 -7.34081429e-05  1.08718136e-01
   7.66391780e-01  8.96971824e-04 -3.13668312e-04  8.73956148e-04
   8.45324786e-07]
 [-3.88830489e-01  6.03081826e-01 -1.07696354e-03  4.10782450e-01
  -5.62452124e-01 -1.80072780e-03 -1.43154206e-03 -1.44792228e-03
  -4.51624906e-06]
 [-1.22843862e-01 -6.04552641e-01 -1.27632206e-03  7.86948023e-01
   1.14319629e-02  1.68573679e-03  1.05935499e-03  1.42011196e-03
   5.11355963e-06]
 [ 1.53887222e-03 -1.60154399e-03 -6.24028587e-01  3.06170479e-04
   2.36553180e-03 -4.55162428e-01 -5.11771883e-01 -3.76153520e-01
  -1.71283609e-03]
 [ 6.03711008e-04  1.49847271e-03 -5.09221831e-01 -1.39858036e-03
  -1.37225472e-03  5.70929537e-01 -3.04104948e-01  5.67667382e-01
   2.55497950e-04]
 [ 2.15610171e-04 -1.10296801e-03  5.88954212e-01  1.01709945e-03
   2.20693647e-05  8.77577787e-02 -8.03326711e-01  9.70975444e-03
   1.06150547e-03]
 [-9.13057997e-06 -2.27837051e-04  6.61743696e-02  8.23529878e-05
   2.02470169e-05 -6.73491742e-01 -1.64028710e-02  7.27985827e-01
  -1.08615601e-01]
 [ 2.43788046e-08 -2.16136157e-05  5.65711864e-03  6.15048843e-06
   3.09205853e-06 -7.46119460e-02 -1.73805364e-03  7.87369544e-02
   9.94081750e-01]]
```

F :

```
F =  [[-1.66142220e-08 -2.16144774e-05  5.65711864e-03]
 [ 6.14723209e-06  3.09199007e-06 -7.46119460e-02]
 [-1.73805365e-03  7.87369544e-02  9.94081750e-01]]
```

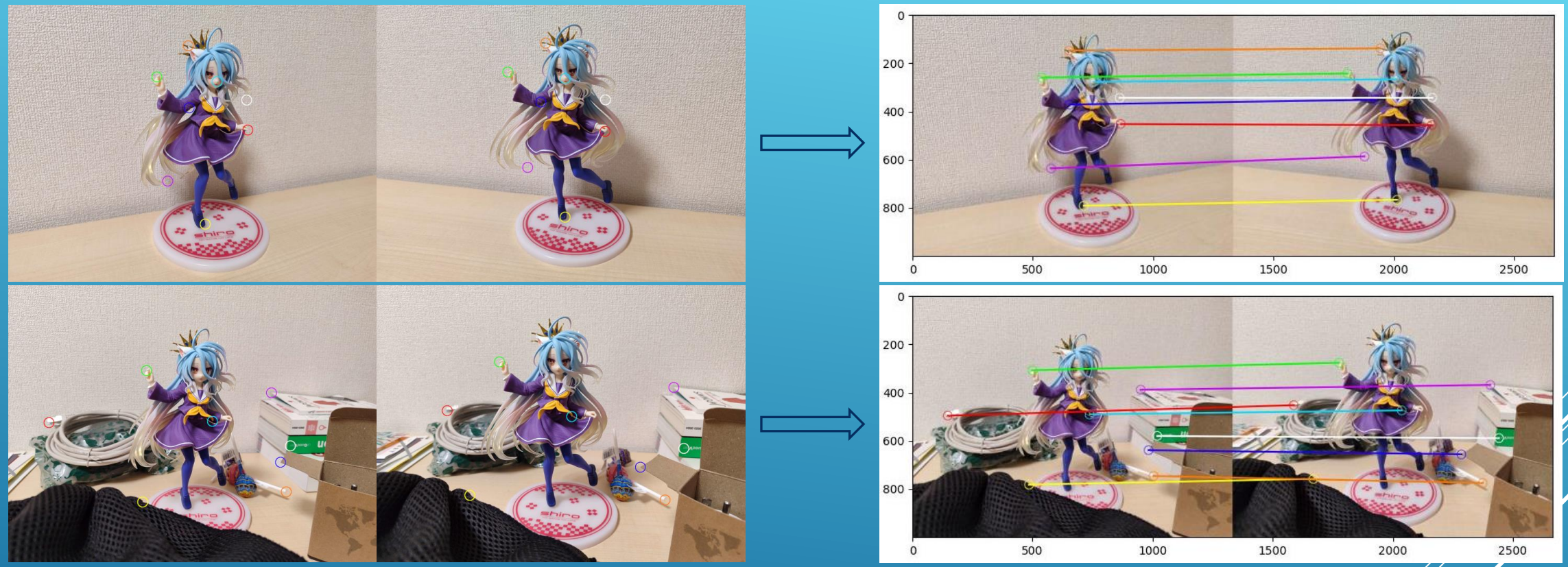
Verification of F :

(with eigenvalues and numpy's rank() function)

```
----------
The eigenvalues of F are :  [9.88126471e-01 9.23
The rank of F is :  2
----------
```

A-2 / A-3 : MATRIX F AND VERIFICATION

Function `concate_img_1_2()` pastes the two images and which draws lines between the corresponding points (recovered previously)



Important: This result image and all images created by the program are saved in the /output folder of each working folder.

A-2 : LINES BETWEEN CORRESPONDING POINTS

A-4 : EPIPOLAR LINES

- We use matplotlib, a Python library for curve plotting
  - Our images in the background
  - Create a space of the same size as our images

  → When we draw a line in our space, it will be an epioplar line in our image

- For each point "i" for which we want an epipolar line:
  - We compute the equation of the line

$$\begin{bmatrix} f_{11}x + f_{21}y + f_{31} & f_{12}x + f_{22}y + f_{32} & f_{13}x + f_{23}y + f_{33} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = 0$$

eq. in img 1 ⟹

```
a1 = F[0][0]*points2[i][0] + F[0][1]*points2[i][1] + F[0][2]
b1 = F[1][0]*points2[i][0] + F[1][1]*points2[i][1] + F[1][2]
c1 = F[2][0]*points2[i][0] + F[2][1]*points2[i][1] + F[2][2]
```

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} f_{11}x' + f_{12}y' + f_{13} \\ f_{21}x' + f_{22}y' + f_{23} \\ f_{31}x' + f_{32}y' + f_{33} \end{bmatrix} = 0$$

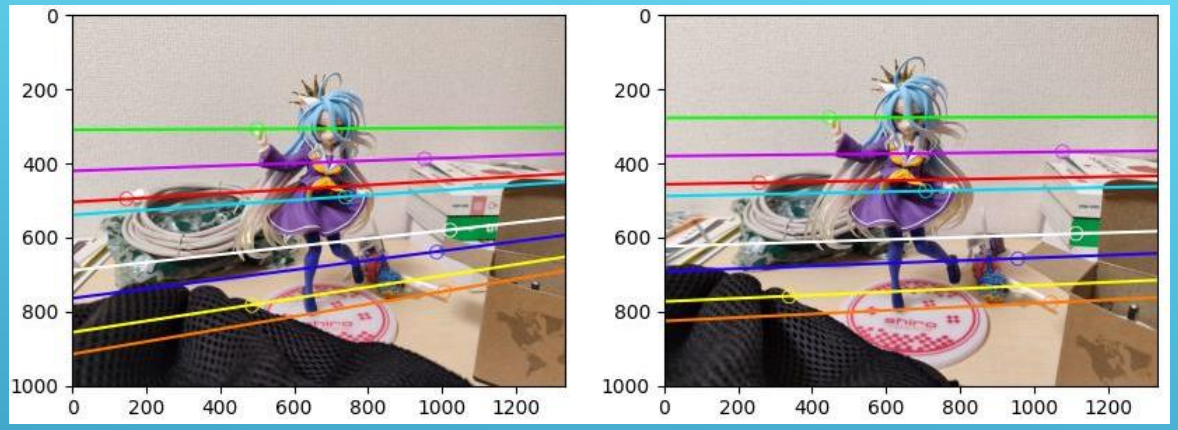
eq. in img 2 ⟹

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

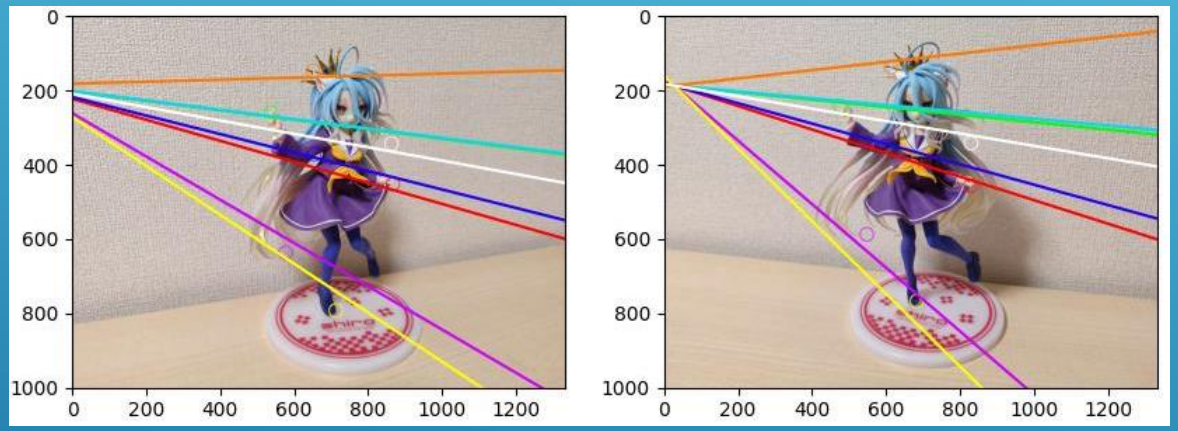
  - Find the equation according to y and draw these lines in both images

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

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

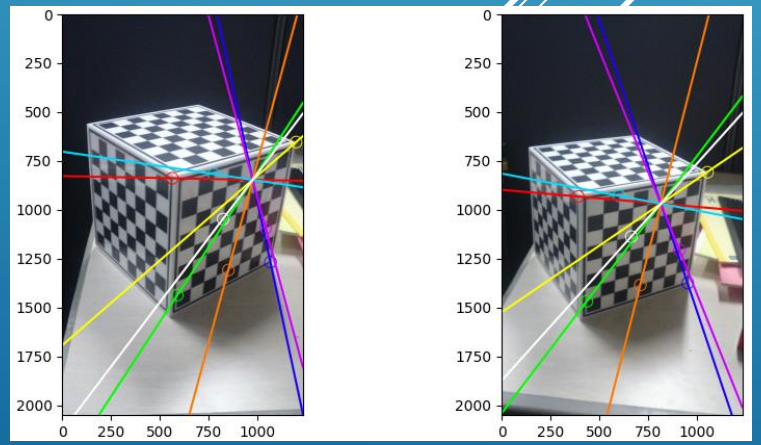


not_same_plane : We notice that the epipolar lines are perfect. We can also note that the lines pass through the center of their circles, so we have good results



same_plane : We notice that the epipolar lines are false and do not pass through the center of the circles, which demonstrates an approximation problem. This probably comes from the quality of the image and the points being too close together (as well as a distortion due to the algorithms of the phone that took the photo). Normalization could help to achieve better results

We see that the problem does not come from the fact that the points are on the same plane, because in this image the epipolar lines are perfect

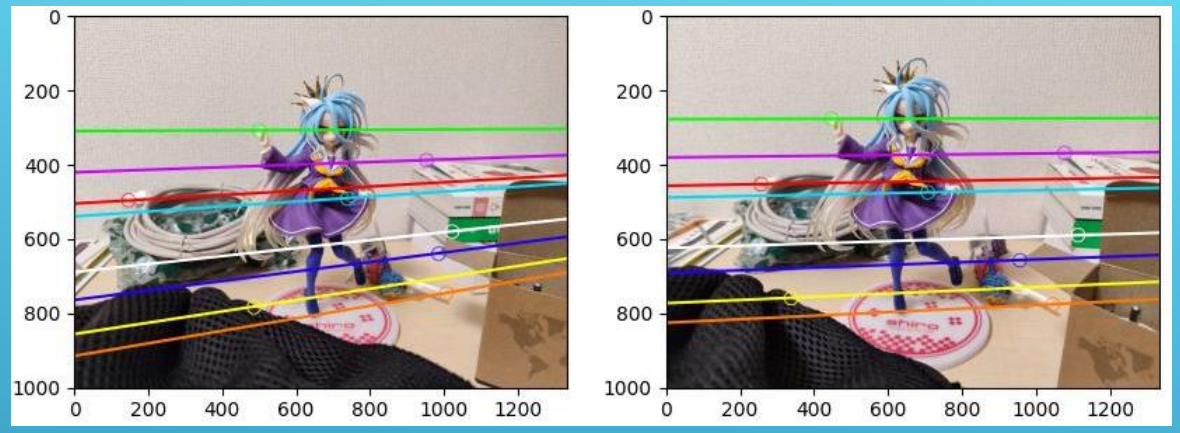
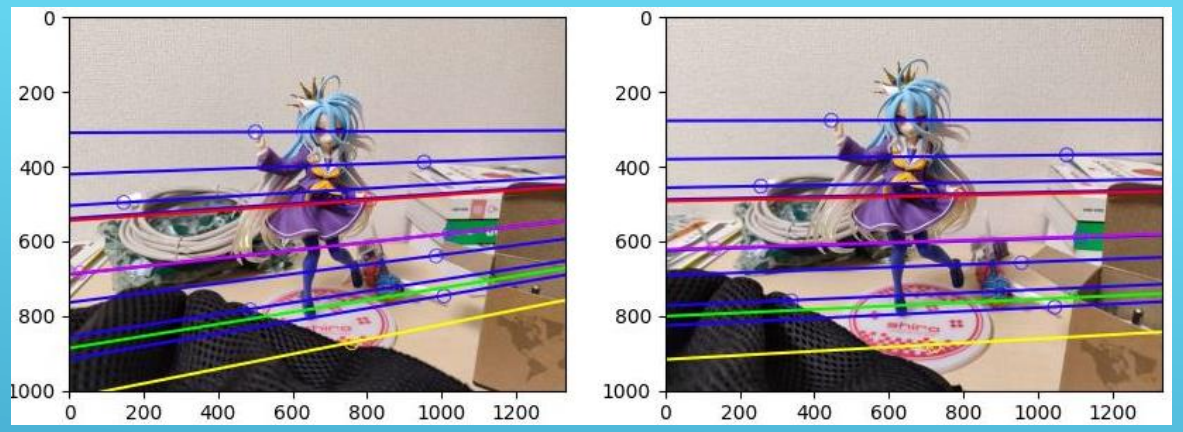
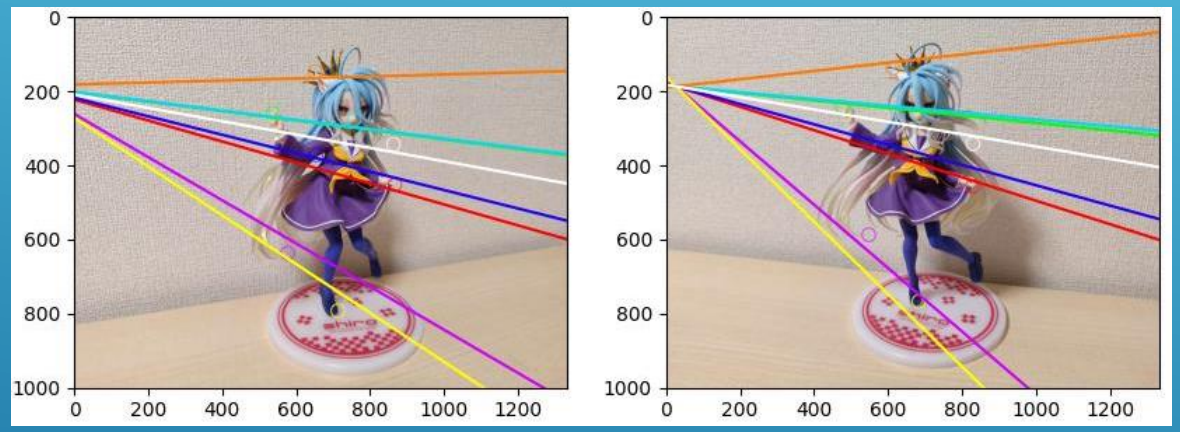
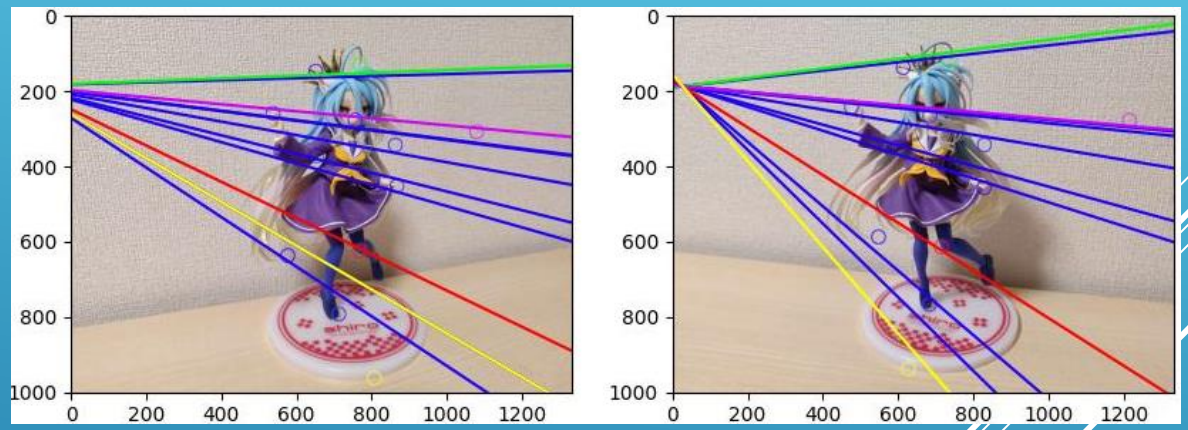


A-4 : RESULTS OF EPIPOLAR LINES

- For each image, we color all the existing circles blue, and we choose new points whose coordinates we put in a new .txt. These new images and this .txt are placed in a /not_computed folder

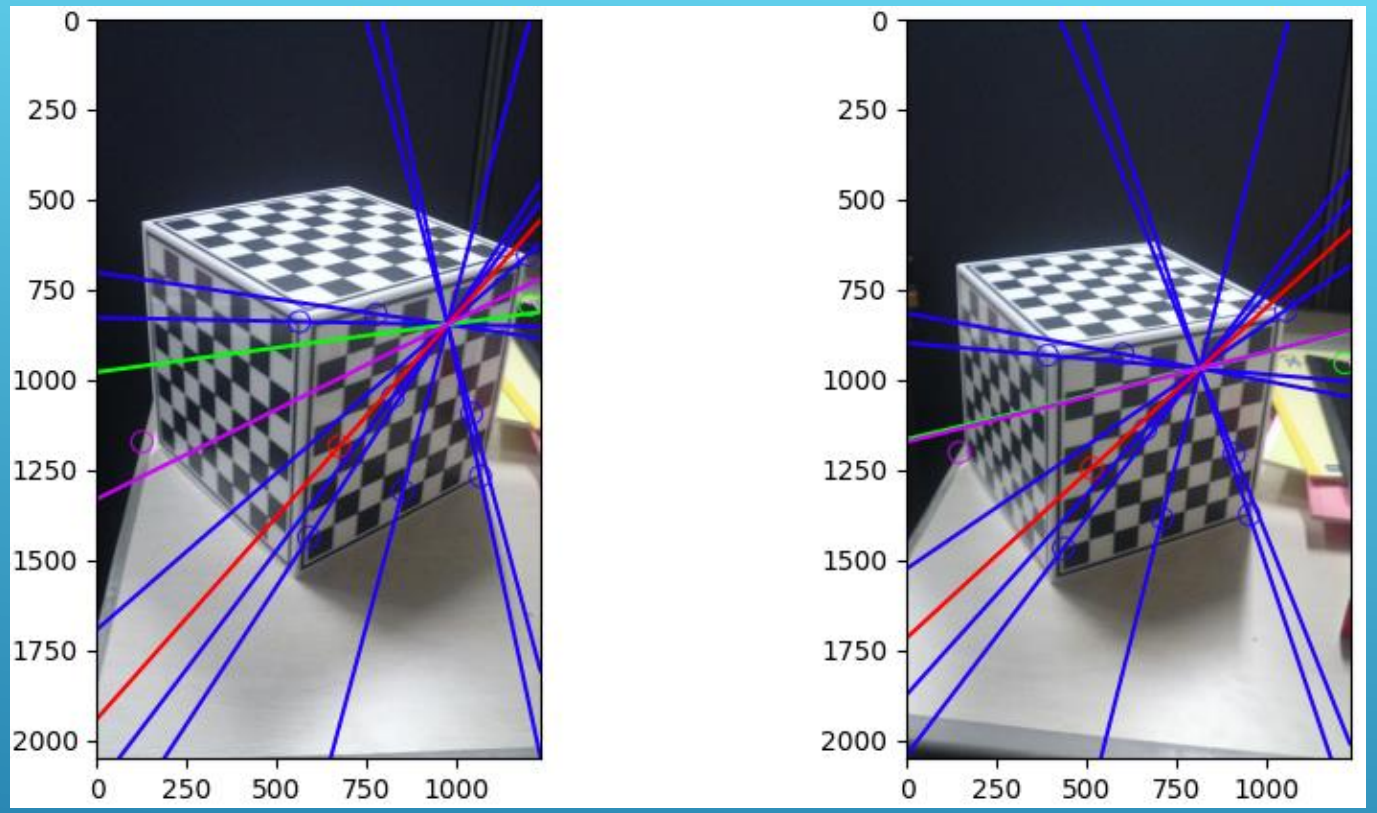
- We open the new images, we redraw the epipolar lines (all in blue this time). Then, we repeat the same algorithm as for A-4 for each new point (which were not used to calculate the matrix F)

A-5 : EPIPOLAR LINES OF NOT COMPUTED POINTS

In `not_same_plane`, we see that the lines are approximately correct.
In `same_plane`, we notice that the red, green and purple points which are on the same plane as the others, have approximately the same divergence. While yellow, much more in the foreground, has a much greater divergence

A-5 : RESULTS OF NOT COMPUTED POINTS



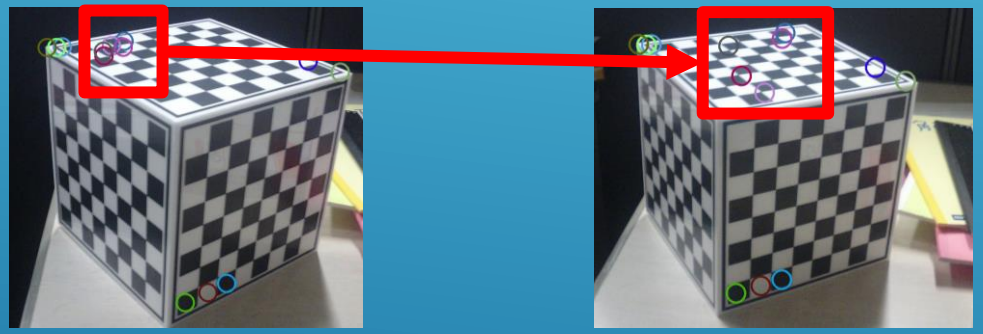
In our "perfect" example (which probably does not have a correction filter in the camera), we notice that the epiopolar line of the red point, on the same plane as the others, is perfect, while we note a slight aberration in the lines of the purple and green points, on a different plane than all the other points which were used to calculate F

We therefore deduce that the matrix F made it possible to calculate the transition from image 1 to image 2, but only on a single plane, and that the data is false if the points sought with this same F are not on this plan

A-5 : RESULTS OF NOT COMPUTED POINTS (PART 2)

- We can no longer use the technique of making circles by hand for this question, because we need to be able to generate many points
- We will therefore make a C++ script (because OpenCV has better mastery of this language) which will generate all these points automatically

- Open images by their names
- Ask the number of feature points that OpenCV must detect and create a matcher with Hamming's distance, which linked points detected in img1 with points in img 2
- Filter the maximal distance of points in img1 and img2 → Very important !



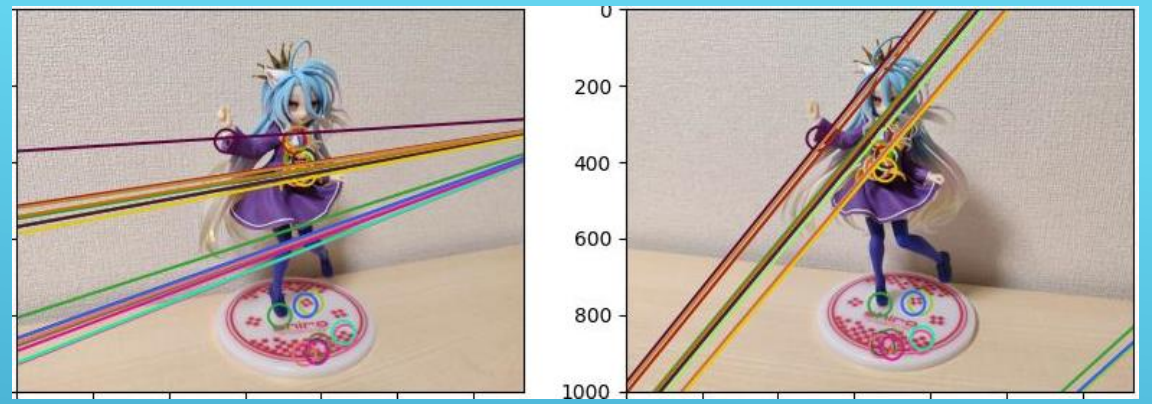
On these images the distance is too high. Filtering allows to remove all points whose distance is too high. However, you must choose the filtering threshold by hand and observe the results

- Once our points are obtained, we save them in the same way like we did before in our .txt file, automatically. We also save the color for each pair of points with which we drew them

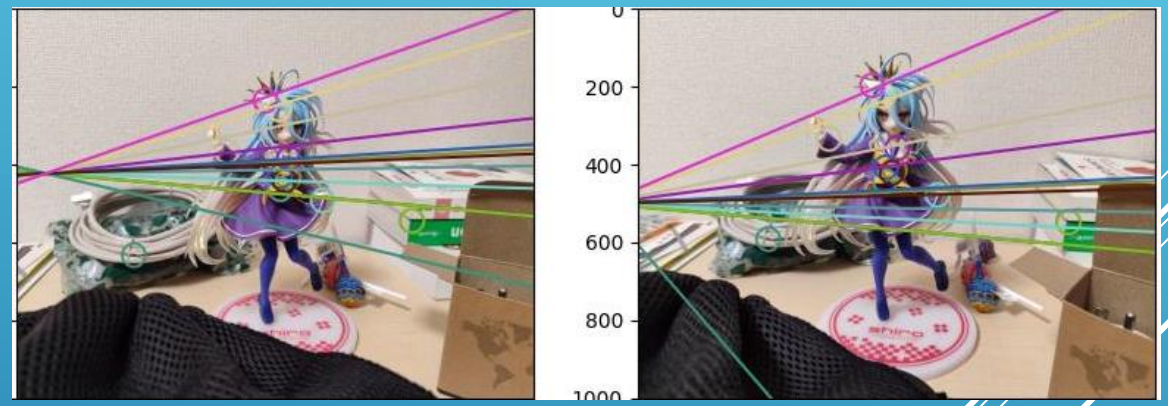
```
img1[ 1152 ; 650 ]/img2[ 1012 ; 802 ]/4bdc5b/
img1[ 166 ; 564 ]/img2[ 171 ; 686 ]/f6ce7f/
img1[ 400 ; 542 ]/img2[ 621 ; 646 ]/346bb0/
img1[ 185 ; 562 ]/img2[ 186 ; 684 ]/4ea0db/
img1[ 1153 ; 650 ]/img2[ 1013 ; 802 ]/86a169/
img1[ 165 ; 564 ]/img2[ 170 ; 685 ]/79ff78/
```

A-6 : F WITH MORE THAN 8 POINTS

23 points



12 points



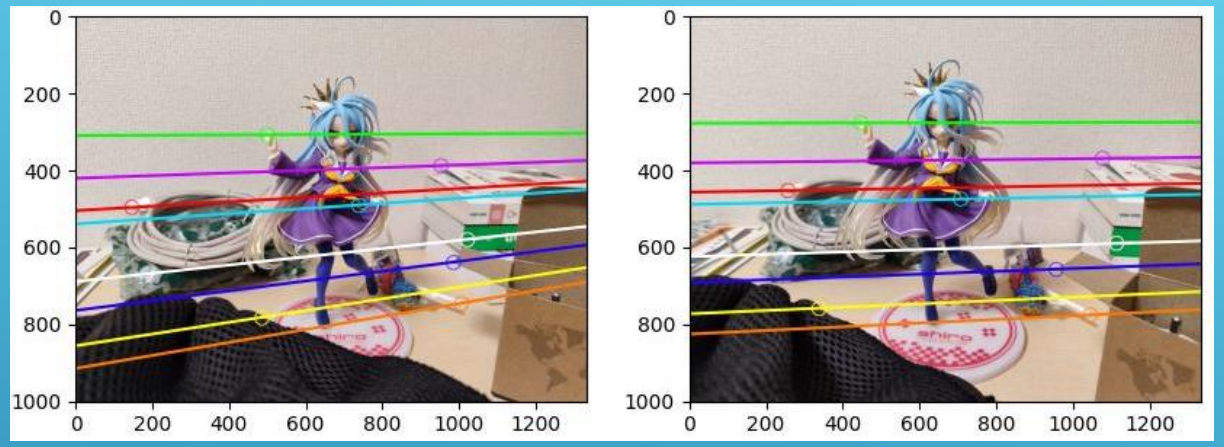
Contrary to what one might think, increasing the number of points does not seem to increase the precision of the matrix F, but on the contrary to significantly reduce it. This can nevertheless come from a precision problem in our C++ program, a distance defect in the OpenCV feature points or even the quality of the image.

A-6 : RESULTS OF MORE THAN 8 POINTS

B-7 : CALCULATE FOCAL LENGTH AND IMAGE CENTER

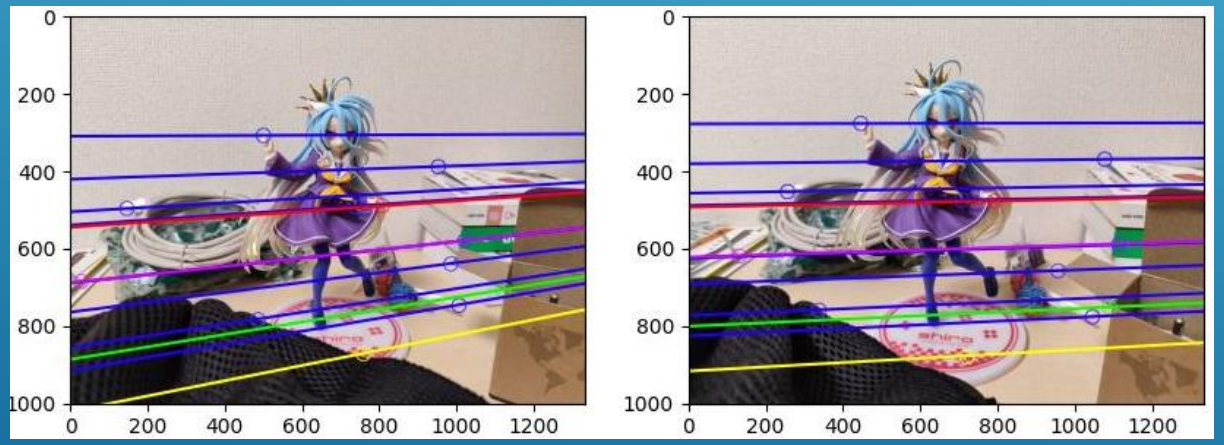
- For each image, we calculate epipoles with the following steps :
  - Decomposition SVD of F^T (for image 1) and F (for image 2)
  - Take the last column of V^T which is the epipole for each image
  - Normalization of the epipole vector
- We need to extend both epipoles into matrices:
  - $e_{1\_mat} = \begin{pmatrix} 0 & e_{1\_z} & e_{1\_y} \\ e_{1\_z} & 0 & -e_{1\_x} \\ -e_{1\_y} & e_{1\_x} & 0 \end{pmatrix}$
  - $e_{2\_mat} = \begin{pmatrix} 0 & e_{2\_z} & e_{2\_y} \\ e_{2\_z} & 0 & -e_{2\_x} \\ -e_{2\_y} & e_{2\_x} & 0 \end{pmatrix}$
- At this point, I don't understand how to use this information and how to extract the focal length and the center point from the F matrix

- Epipolar lines



Epipolar lines are very precise in cases where the points are quite far apart and quite precise. We also saw that the quality of the image could impact the result, as well as digital processing when the photo is taken by phones. We also saw that taking points on the same plane or on different planes did not seem to have a major impact on the results.

- Epipolar lines with not computed points

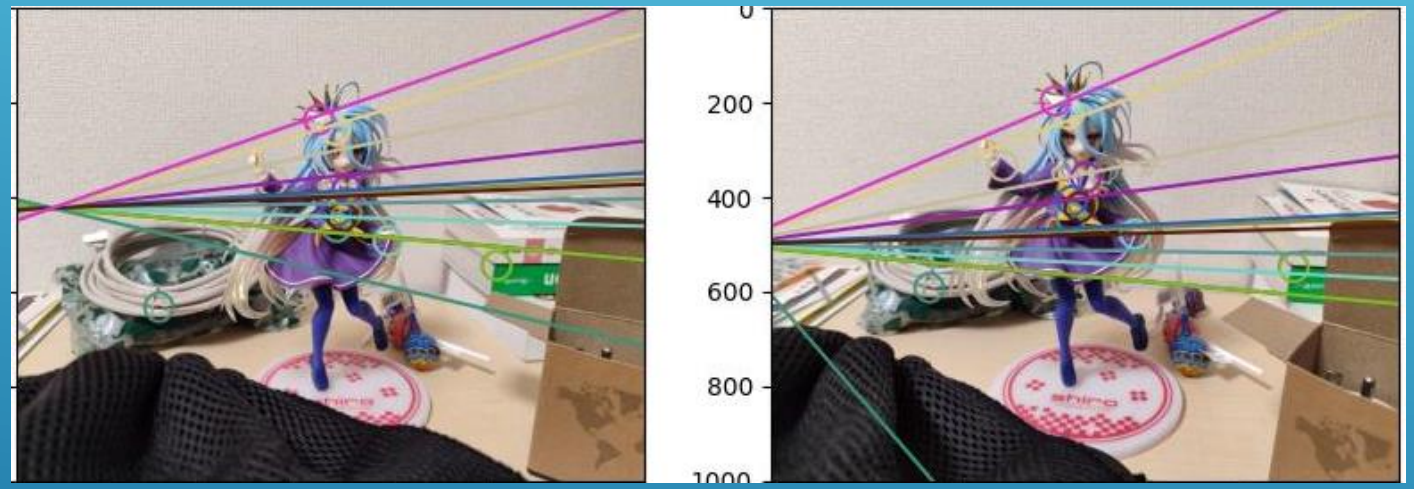


The epipolar lines of the points which were not used to calculate the matrix F are quite variable. Indeed, when F has been calculated with points on the same plane, then taking new points which are not on this plane seems to cause errors. However, the lines are much more precise when the points that calculated F come from different planes. We still note errors when the points are too far from the initial points.

B-9 : SUMMARY

B-9 : SUMMARY (PART 2)

- More than 8 points



When we calculate the matrix F with more than 8 points, we often arrive at inconsistent results. Indeed, half of the points are quite good, while we see huge errors on another half. This can come from OpenCV feature points which are not pixel-precise, but also from the quality of the points chosen: the algorithm used matters a lot, and if most of the points are grouped in the same place, this can explain the poor results observed.