HACETTEPE UNIVERSITY DEPARTMENT OF GEOMATICS ENGINEERING

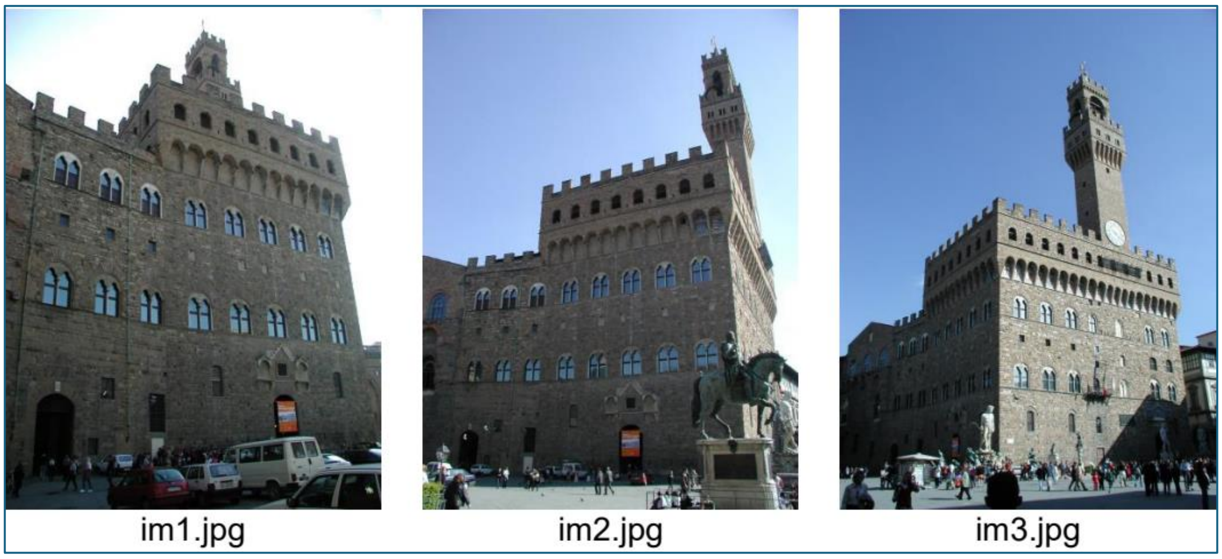
GMT431- PHOTOGRAMMETRIC IMAGE ANALYSIS

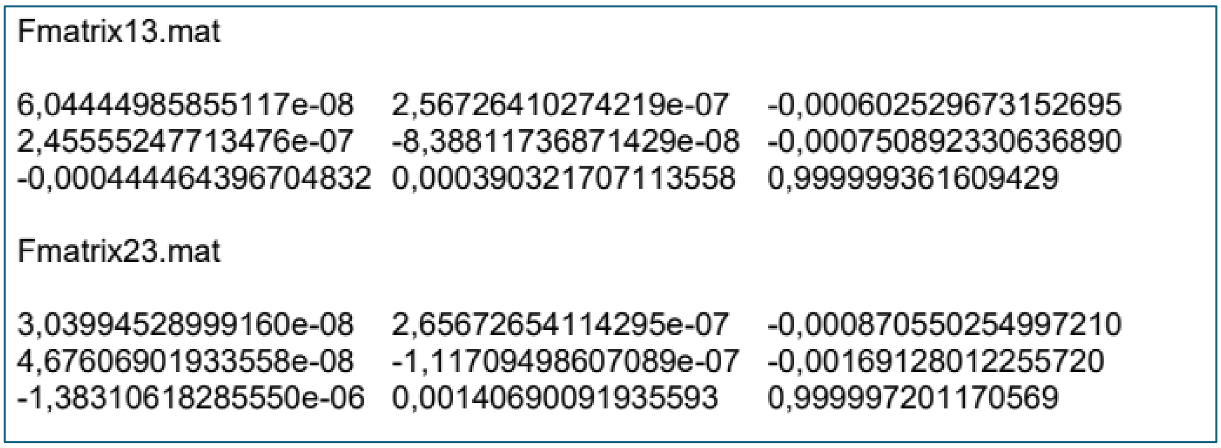
ASSIGNMENT-2 REPORT

Egehan YAĞLICI

21967847

First of all, for this task we have 3 images called florence1, florence2 and florence3 and 2 basis matrices. These matrices are Fmatrix13 for florence1 and florence3 and Fmatrix23 for florence2 and florence3.





```
Fmatrix13.mat

6,04444985855117e-08    2,56726410274219e-07    -0,000602529673152695
2,45555247713476e-07    -8,38811736871429e-08   -0,000750892330636890
-0,000444464396704832   0,000390321707113558    0,999999361609429

Fmatrix23.mat

3,03994528999160e-08    2,65672654114295e-07    -0,000870550254997210
4,67606901933558e-08    -1,11709498607089e-07   -0,00169128012255720
-1,38310618285550e-06   0,00140690091935593     0,999997201170569
```

We have to select five points from the florence2 image for the first question. Following this, we must use the fundamental matrix to locate the epipolar lines on the florence3 image.

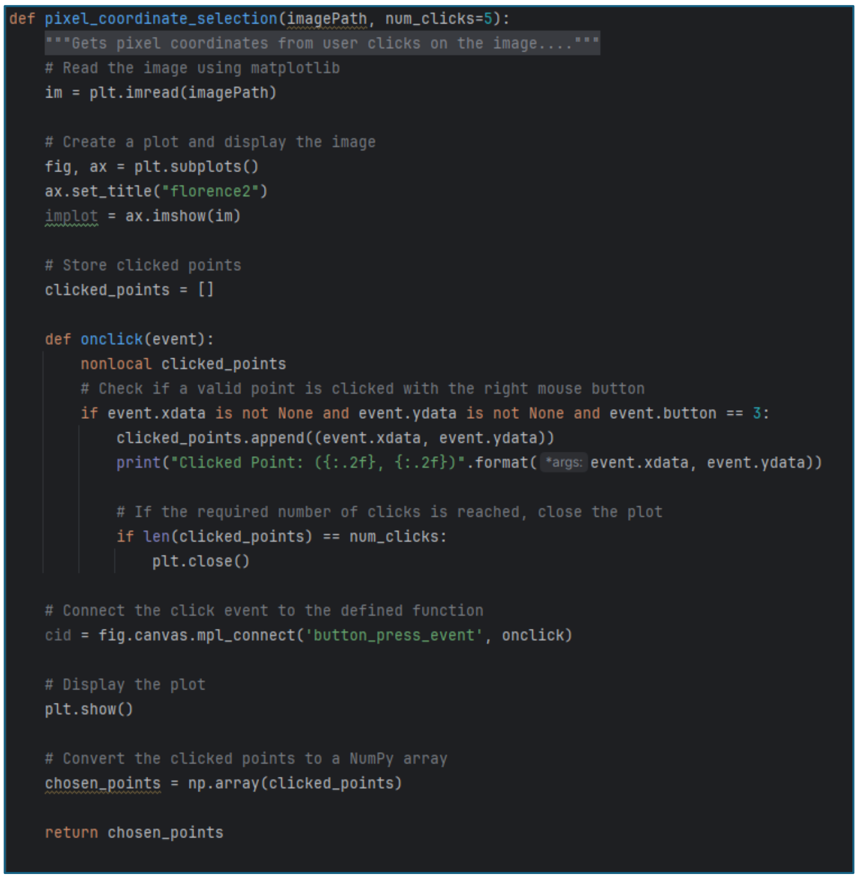
Next, we must choose five points from each of the florence1 and florence2 images that correspond to the same locations in the image. We must subsequently process the fundamental matrices using these points to find the intersection points of the epipolar lines on the florence3 image.

Let’s start with first question. I recommend that you run the code with Pycharm because interactive parts may not work in other sources. That's why I made a time-lapse video for two questions.

There are also explanations in the code, but I have closed the explanations and put them here because they take up space. Also, yes, I proceeded by writing a function for everything, just like in the last assignment :).

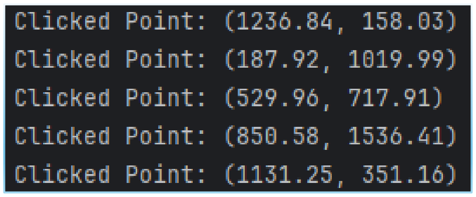
Q1) You are required to write a code that loads the image pair 2-3 and use the provided fundamental matrix (Fmatrix23) to compute and display the corresponding epipolar lines for manually selected points. (i.e. select manually 5 points in im2 and display the corresponding epipolar lines (with different line colors) on im3. Validate visually that the epipolar lines go through the selected points in the other image.)

I intended to create an interactive code for this question that would allow the user to select the spot they chose. In response to the second question, I upgraded the same feature.



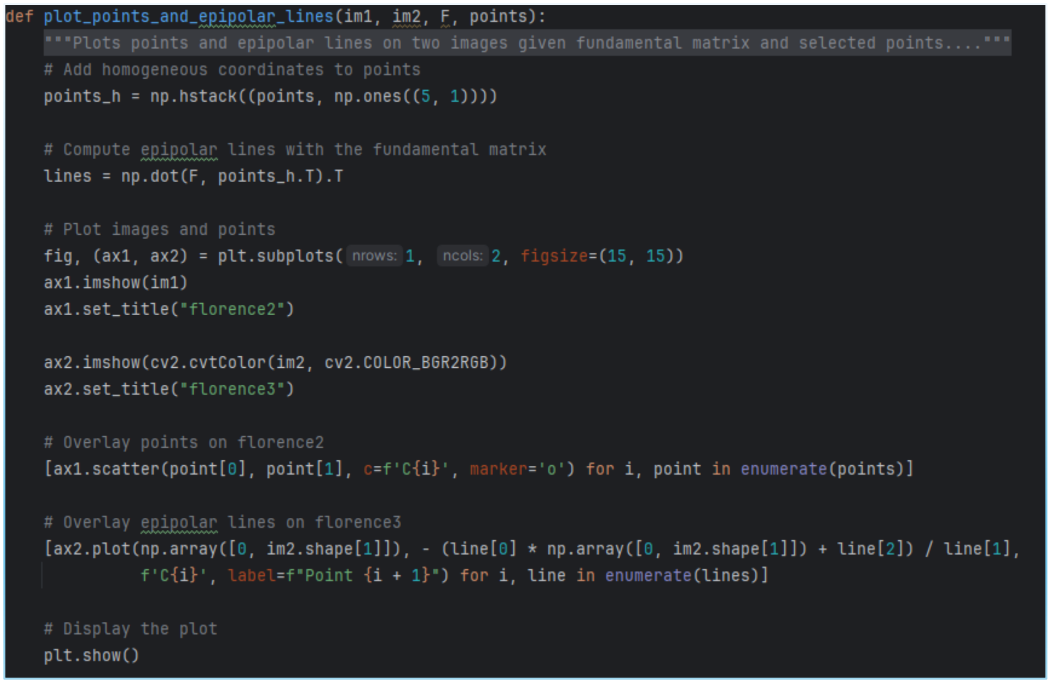
```python
def pixel_coordinate_selection(imagePath, num_clicks=5):
    """Gets pixel coordinates from user clicks on the image...."""
    # Read the image using matplotlib
    im = plt.imread(imagePath)

    # Create a plot and display the image
    fig, ax = plt.subplots()
    ax.set_title("florence2")
    implot = ax.imshow(im)

    # Store clicked points
    clicked_points = []

    def onclick(event):
        nonlocal clicked_points
        # Check if a valid point is clicked with the right mouse button
        if event.xdata is not None and event.ydata is not None and event.button == 3:
            clicked_points.append((event.xdata, event.ydata))
            print("Clicked Point: ({:.2f}, {:.2f})".format( *args: event.xdata, event.ydata))

            # If the required number of clicks is reached, close the plot
            if len(clicked_points) == num_clicks:
                plt.close()

    # Connect the click event to the defined function
    cid = fig.canvas.mpl_connect('button_press_event', onclick)

    # Display the plot
    plt.show()

    # Convert the clicked points to a NumPy array
    chosen_points = np.array(clicked_points)

    return chosen_points
```

As can be seen from the image on the left, the function named "pixel_coordinate_selection" opens a window to the user and allows the user to select 5 points using the right mouse button. The reason why this is the right button is that possible zoom operations performed by the user with the left mouse button may accidentally trigger the point selection process.



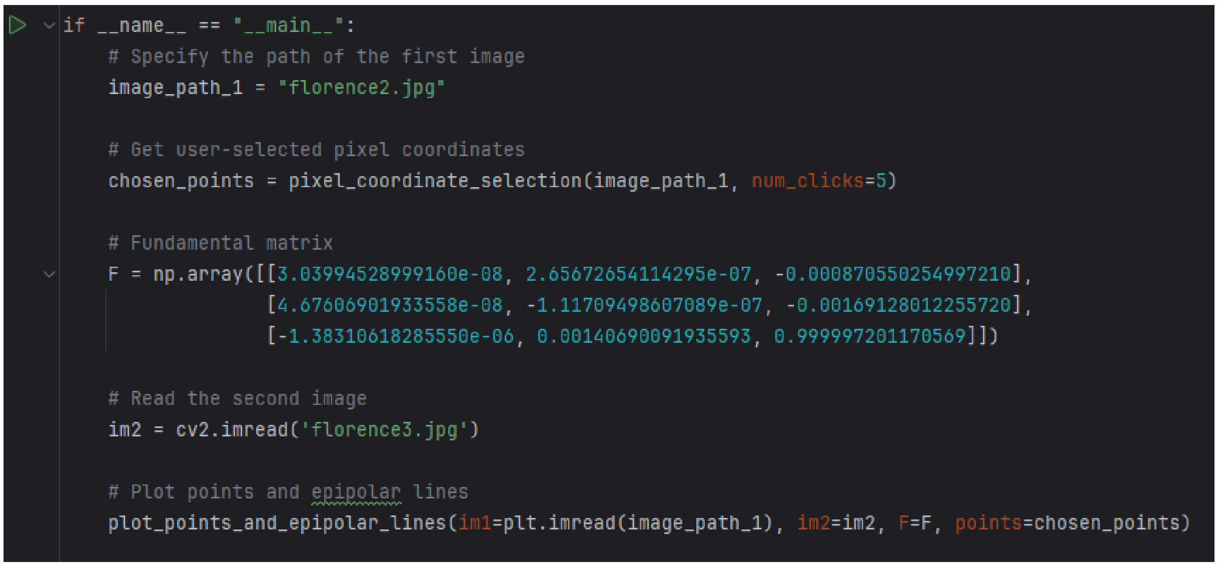
```
Clicked Point: (1236.84, 158.03)
Clicked Point: (187.92, 1019.99)
Clicked Point: (529.96, 717.91)
Clicked Point: (850.58, 1536.41)
Clicked Point: (1131.25, 351.16)
```

It prints each selected point with "Clicked Point" output. These points are stored as an array variable named chosen_points for epipolar line calculation.



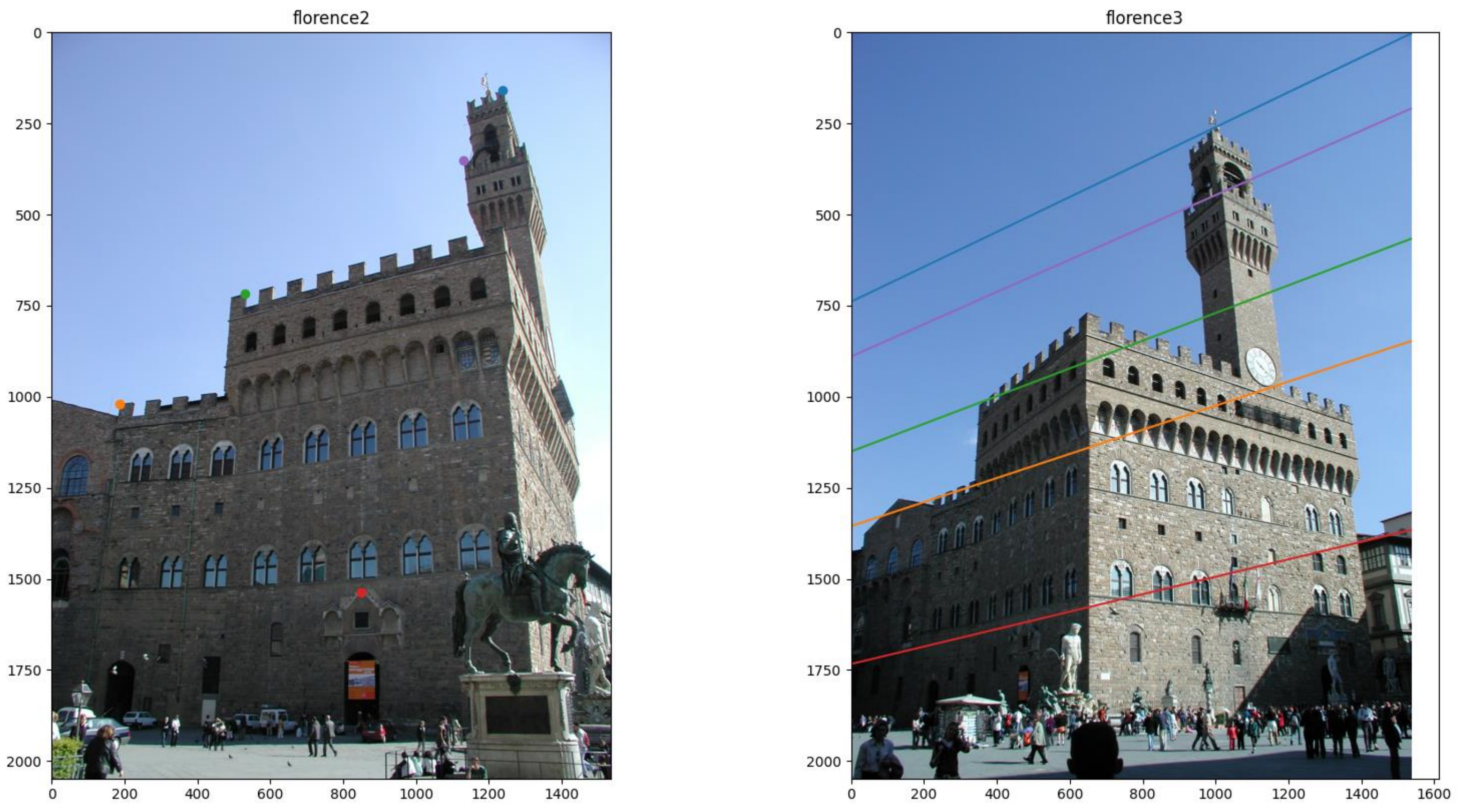
```python
def plot_points_and_epipolar_lines(im1, im2, F, points):
    """Plots points and epipolar lines on two images given fundamental matrix and selected points...."""
    # Add homogeneous coordinates to points
    points_h = np.hstack((points, np.ones((5, 1))))

    # Compute epipolar lines with the fundamental matrix
    lines = np.dot(F, points_h.T).T

    # Plot images and points
    fig, (ax1, ax2) = plt.subplots( nrows: 1,  ncols: 2, figsize=(15, 15))
    ax1.imshow(im1)
    ax1.set_title("florence2")

    ax2.imshow(cv2.cvtColor(im2, cv2.COLOR_BGR2RGB))
    ax2.set_title("florence3")

    # Overlay points on florence2
    [ax1.scatter(point[0], point[1], c=f'C{i}', marker='o') for i, point in enumerate(points)]

    # Overlay epipolar lines on florence3
    [ax2.plot(np.array([0, im2.shape[1]]), - (line[0] * np.array([0, im2.shape[1]]) + line[2]) / line[1],
              f'C{i}', label=f"Point {i + 1}") for i, line in enumerate(lines)]

    # Display the plot
    plt.show()
```

With the pixel coordinates we have, the plot_points_and_epipolar_lines function marks the chosen points on the florence2 image. It then adds a dimension to work in a homogeneous environment, multiplies (dot product) the fundamental matrix by the transpose of the pixel coordinates, and transposes the outcome to change the matrix from column to row and identify the epipolar lines. It creates epipolar lines on the florence3 image as a result of the operation.



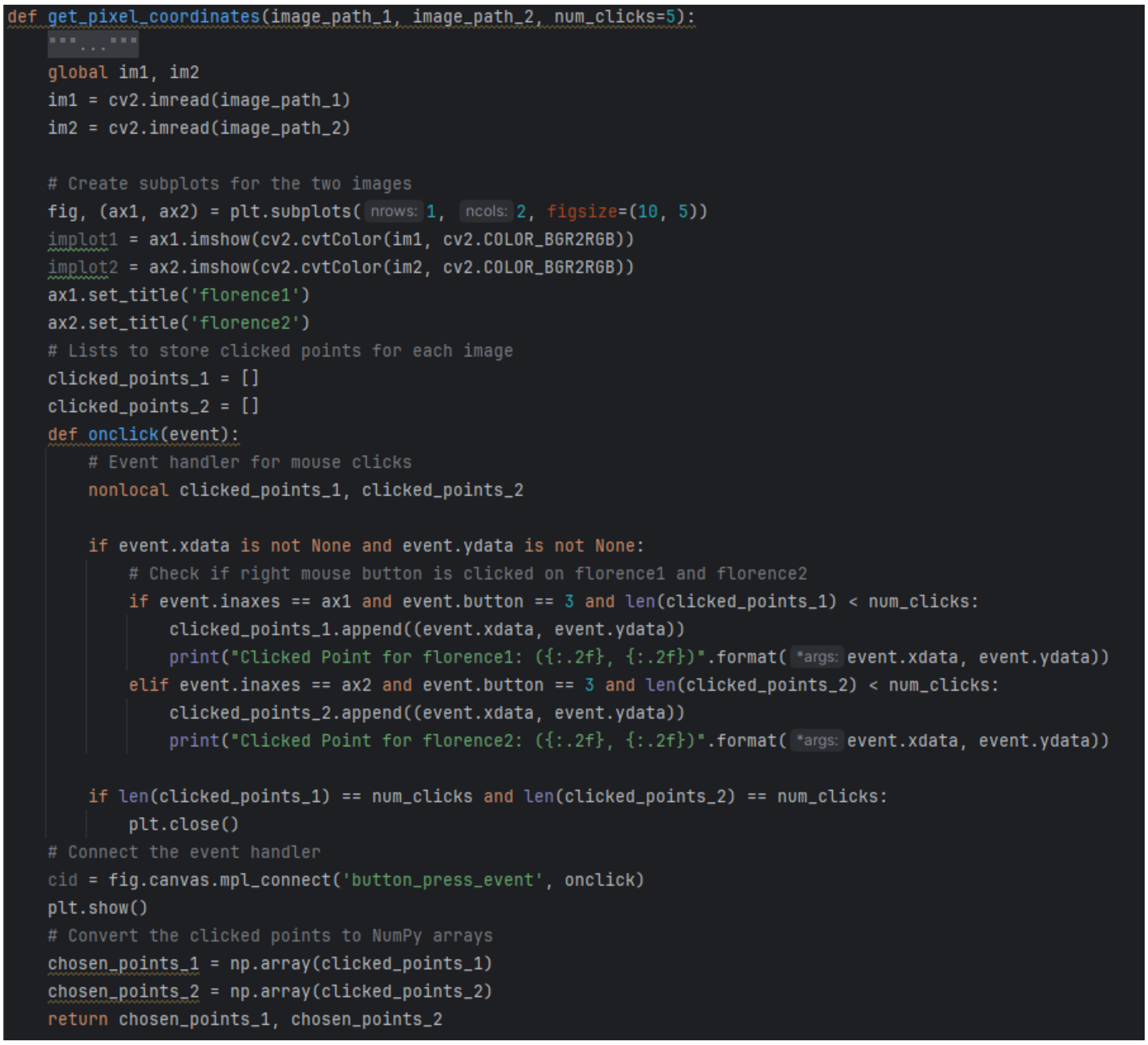
```python
if __name__ == "__main__":
    # Specify the path of the first image
    image_path_1 = "florence2.jpg"

    # Get user-selected pixel coordinates
    chosen_points = pixel_coordinate_selection(image_path_1, num_clicks=5)

    # Fundamental matrix
    F = np.array([[3.03994528999160e-08, 2.65672654114295e-07, -0.000870550254997210],
                  [4.67606901933558e-08, -1.11709498607089e-07, -0.00169128012255720],
                  [-1.38310618285550e-06, 0.00140690091935593, 0.999997201170569]])

    # Read the second image
    im2 = cv2.imread('florence3.jpg')

    # Plot points and epipolar lines
    plot_points_and_epipolar_lines(im1=plt.imread(image_path_1), im2=im2, F=F, points=chosen_points)
```

After writing the functions, we place the required image and fundamental matrix and run the code. If you put florence images in the same working environment, you can use them without writing a path.

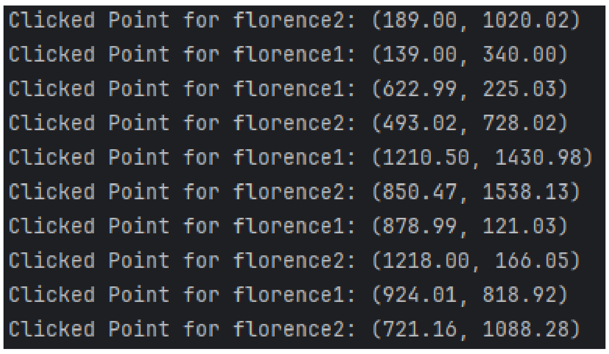


Q2) Manually select 5 corresponding points in image pair 1-2 and find a way to compute the projection of these points on the third image. (i.e. select 5 points in im1 and find and select the same point locations in im2. Transfer these points on im3 using epipolar geometry.)

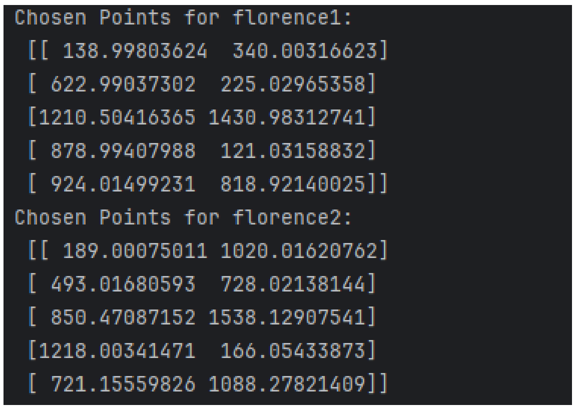
I added a slightly improved version of the function that enables selection with the mouse to this code. Here, for you to select a point, it opens two images in a single window on the screen and allows you to select 5 points on each image, and whichever point you select, it prints which point and its pixel coordinates for which image to the console. It does not allow the user to select more than 5 points in a single image. After the selection it saves all the points in arrays and print final versions of them. Right click select!



```
def get_pixel_coordinates(image_path_1, image_path_2, num_clicks=5):
    """..."""
    global im1, im2
    im1 = cv2.imread(image_path_1)
    im2 = cv2.imread(image_path_2)

    # Create subplots for the two images
    fig, (ax1, ax2) = plt.subplots( nrows: 1,  ncols: 2, figsize=(10, 5))
    implot1 = ax1.imshow(cv2.cvtColor(im1, cv2.COLOR_BGR2RGB))
    implot2 = ax2.imshow(cv2.cvtColor(im2, cv2.COLOR_BGR2RGB))
    ax1.set_title('florence1')
    ax2.set_title('florence2')
    # Lists to store clicked points for each image
    clicked_points_1 = []
    clicked_points_2 = []
    def onclick(event):
        # Event handler for mouse clicks
        nonlocal clicked_points_1, clicked_points_2

        if event.xdata is not None and event.ydata is not None:
            # Check if right mouse button is clicked on florence1 and florence2
            if event.inaxes == ax1 and event.button == 3 and len(clicked_points_1) < num_clicks:
                clicked_points_1.append((event.xdata, event.ydata))
                print("Clicked Point for florence1: ({:.2f}, {:.2f})".format( *args: event.xdata, event.ydata))
            elif event.inaxes == ax2 and event.button == 3 and len(clicked_points_2) < num_clicks:
                clicked_points_2.append((event.xdata, event.ydata))
                print("Clicked Point for florence2: ({:.2f}, {:.2f})".format( *args: event.xdata, event.ydata))

        if len(clicked_points_1) == num_clicks and len(clicked_points_2) == num_clicks:
            plt.close()
    # Connect the event handler
    cid = fig.canvas.mpl_connect('button_press_event', onclick)
    plt.show()
    # Convert the clicked points to NumPy arrays
    chosen_points_1 = np.array(clicked_points_1)
    chosen_points_2 = np.array(clicked_points_2)
    return chosen_points_1, chosen_points_2
```



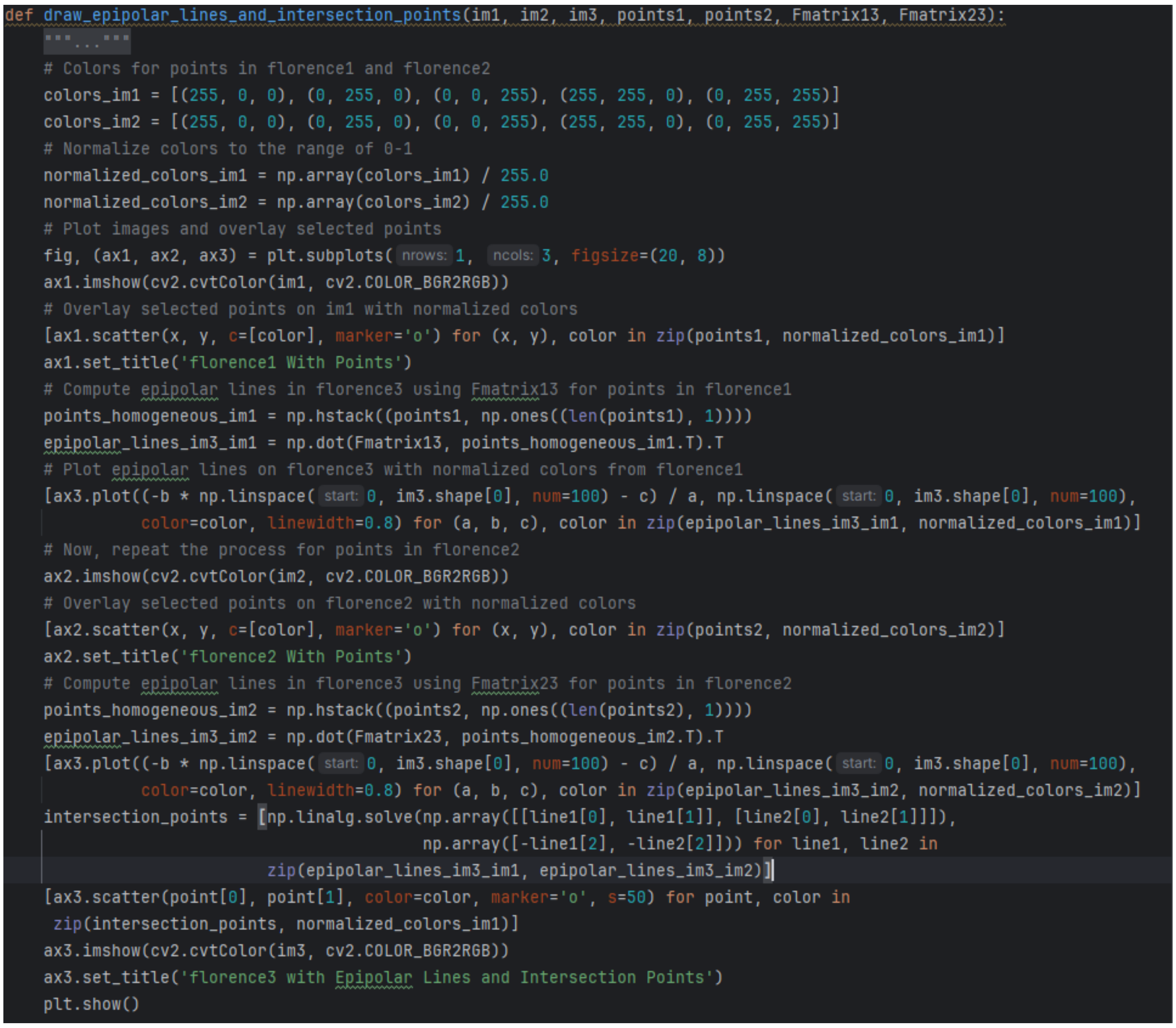
```
Clicked Point for florence2: (189.00, 1020.02)
Clicked Point for florence1: (139.00, 340.00)
Clicked Point for florence1: (622.99, 225.03)
Clicked Point for florence2: (493.02, 728.02)
Clicked Point for florence1: (1210.50, 1430.98)
Clicked Point for florence2: (850.47, 1538.13)
Clicked Point for florence1: (878.99, 121.03)
Clicked Point for florence2: (1218.00, 166.05)
Clicked Point for florence1: (924.01, 818.92)
Clicked Point for florence2: (721.16, 1088.28)
```



```
Chosen Points for florence1:
 [[ 138.99803624  340.00316623]
 [ 622.99037302  225.02965358]
 [1210.50416365 1430.98312741]
 [ 878.99407988  121.03158832]
 [ 924.01499231  818.92140025]]
Chosen Points for florence2:
 [[ 189.00075011 1020.01620762]
 [ 493.01680593  728.02138144]
 [ 850.47087152 1538.12907541]
 [1218.00341471  166.05433873]
 [ 721.15559826 1088.27821409]]
```

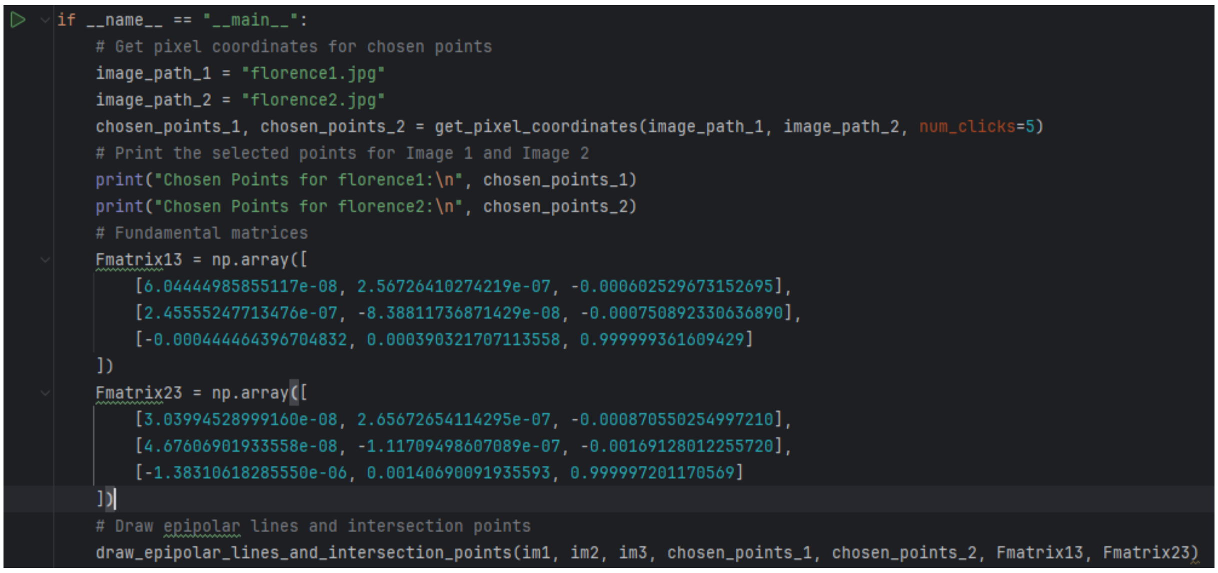
The draw_epipolar_lines_and_intersection_points function selects a point at a specific location and applies the same color to it. The selected points color is also associated with epipolar lines and intersections. In the final image, everything is more visible.

This function's basic operation follows the same logic as the one used in the first question, inserting the chosen points using the dot product operation and fundamental matrices to find the epipolar lines (ax + by + c). On the florence3 image, it displays the determined lines and the spots where they connect.



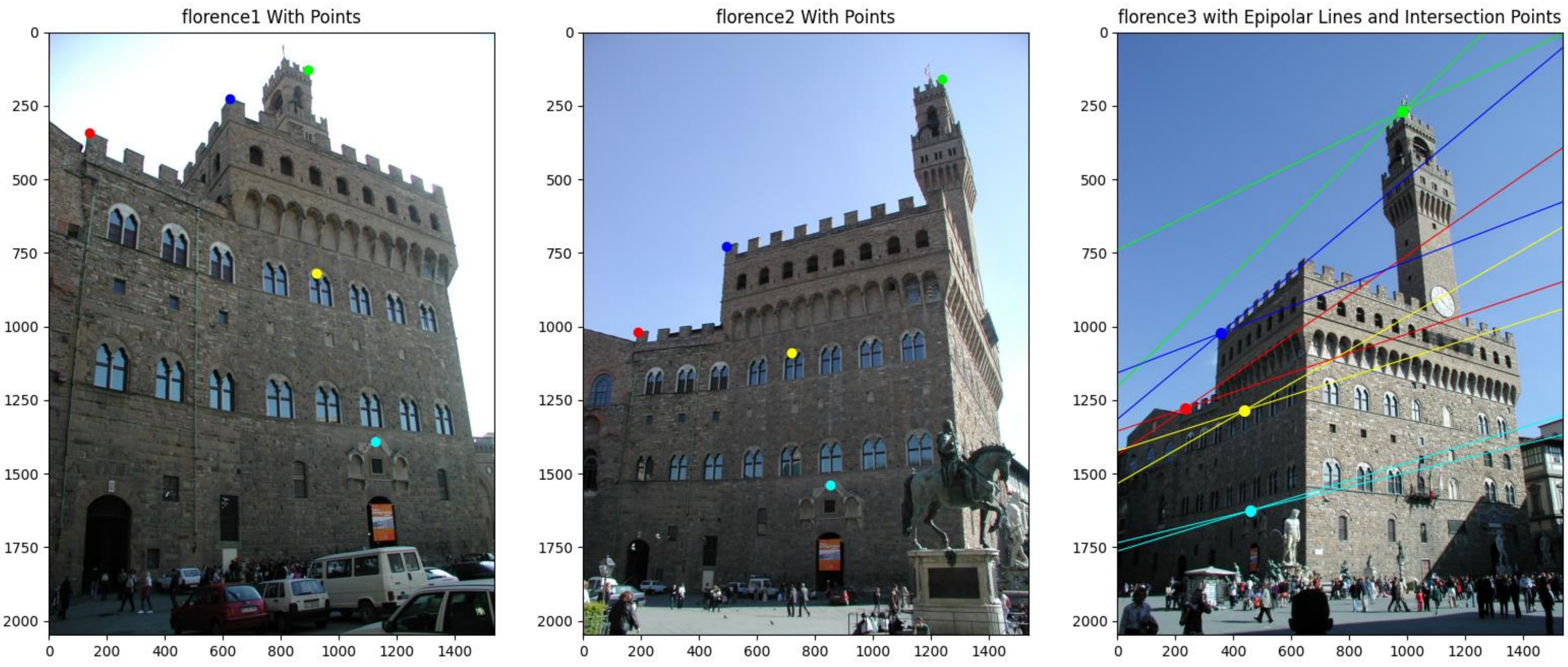
```python
def draw_epipolar_lines_and_intersection_points(im1, im2, im3, points1, points2, Fmatrix13, Fmatrix23):
    """..."""
    # Colors for points in florence1 and florence2
    colors_im1 = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0, 255, 255)]
    colors_im2 = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0), (0, 255, 255)]
    # Normalize colors to the range of 0-1
    normalized_colors_im1 = np.array(colors_im1) / 255.0
    normalized_colors_im2 = np.array(colors_im2) / 255.0
    # Plot images and overlay selected points
    fig, (ax1, ax2, ax3) = plt.subplots( nrows: 1,  ncols: 3, figsize=(20, 8))
    ax1.imshow(cv2.cvtColor(im1, cv2.COLOR_BGR2RGB))
    # Overlay selected points on im1 with normalized colors
    [ax1.scatter(x, y, c=[color], marker='o') for (x, y), color in zip(points1, normalized_colors_im1)]
    ax1.set_title('florence1 With Points')
    # Compute epipolar lines in florence3 using Fmatrix13 for points in florence1
    points_homogeneous_im1 = np.hstack((points1, np.ones((len(points1), 1))))
    epipolar_lines_im3_im1 = np.dot(Fmatrix13, points_homogeneous_im1.T).T
    # Plot epipolar lines on florence3 with normalized colors from florence1
    [ax3.plot((-b * np.linspace( start: 0, im3.shape[0], num=100) - c) / a, np.linspace( start: 0, im3.shape[0], num=100),
              color=color, linewidth=0.8) for (a, b, c), color in zip(epipolar_lines_im3_im1, normalized_colors_im1)]
    # Now, repeat the process for points in florence2
    ax2.imshow(cv2.cvtColor(im2, cv2.COLOR_BGR2RGB))
    # Overlay selected points on florence2 with normalized colors
    [ax2.scatter(x, y, c=[color], marker='o') for (x, y), color in zip(points2, normalized_colors_im2)]
    ax2.set_title('florence2 With Points')
    # Compute epipolar lines in florence3 using Fmatrix23 for points in florence2
    points_homogeneous_im2 = np.hstack((points2, np.ones((len(points2), 1))))
    epipolar_lines_im3_im2 = np.dot(Fmatrix23, points_homogeneous_im2.T).T
    [ax3.plot((-b * np.linspace( start: 0, im3.shape[0], num=100) - c) / a, np.linspace( start: 0, im3.shape[0], num=100),
              color=color, linewidth=0.8) for (a, b, c), color in zip(epipolar_lines_im3_im2, normalized_colors_im2)]
    intersection_points = [np.linalg.solve(np.array([[line1[0], line1[1]], [line2[0], line2[1]]]),
                                           np.array([-line1[2], -line2[2]])) for line1, line2 in
                           zip(epipolar_lines_im3_im1, epipolar_lines_im3_im2)]
    [ax3.scatter(point[0], point[1], color=color, marker='o', s=50) for point, color in
     zip(intersection_points, normalized_colors_im1)]
    ax3.imshow(cv2.cvtColor(im3, cv2.COLOR_BGR2RGB))
    ax3.set_title('florence3 with Epipolar Lines and Intersection Points')
    plt.show()
```

After writing these two functions, we run the code by placing our visual path and fundamental matrices.

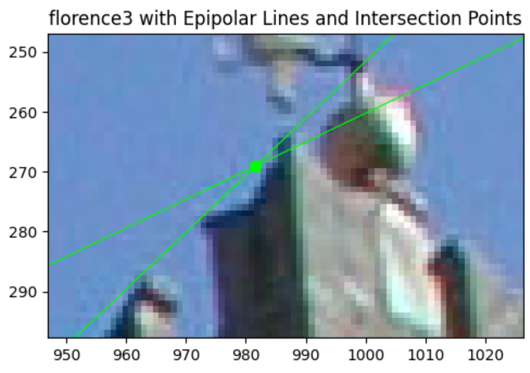


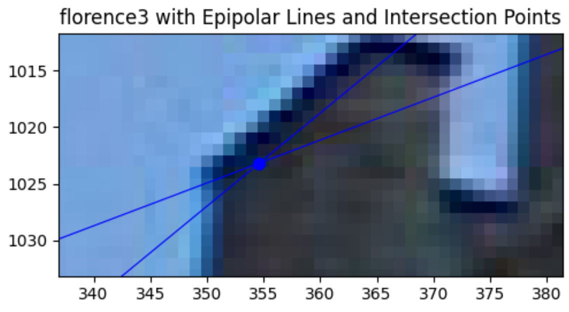
```python
if __name__ == "__main__":
    # Get pixel coordinates for chosen points
    image_path_1 = "florence1.jpg"
    image_path_2 = "florence2.jpg"
    chosen_points_1, chosen_points_2 = get_pixel_coordinates(image_path_1, image_path_2, num_clicks=5)
    # Print the selected points for Image 1 and Image 2
    print("Chosen Points for florence1:\n", chosen_points_1)
    print("Chosen Points for florence2:\n", chosen_points_2)
    # Fundamental matrices
    Fmatrix13 = np.array([
        [6.04444985855117e-08, 2.56726410274219e-07, -0.000602529673152695],
        [2.45555247713476e-07, -8.38811736871429e-08, -0.000750892330636890],
        [-0.000444464396704832, 0.000390321707113558, 0.999999361609429]
    ])
    Fmatrix23 = np.array([
        [3.03994528999160e-08, 2.65672654114295e-07, -0.000870550254997210],
        [4.67606901933558e-08, -1.11709498607089e-07, -0.00169128012255720],
        [-1.38310618285550e-06, 0.00140690091935593, 0.999997201170569]
    ])
    # Draw epipolar lines and intersection points
    draw_epipolar_lines_and_intersection_points(im1, im2, im3, chosen_points_1, chosen_points_2, Fmatrix13, Fmatrix23)
```

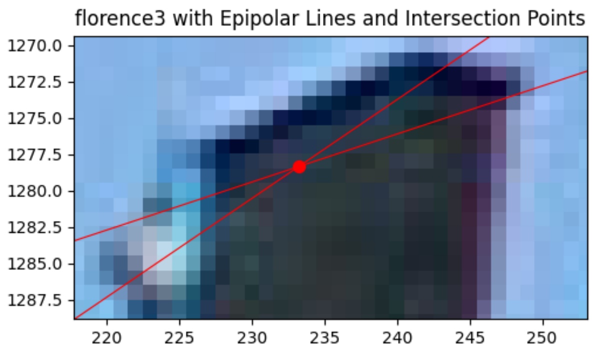
The final result is.

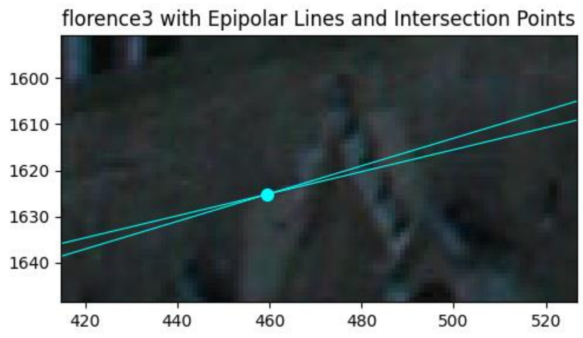


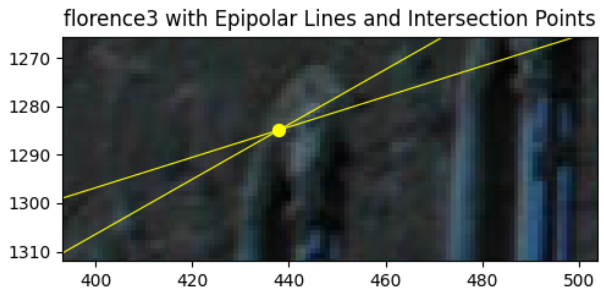
But… there is some errors as we can see, so why we got these errors.









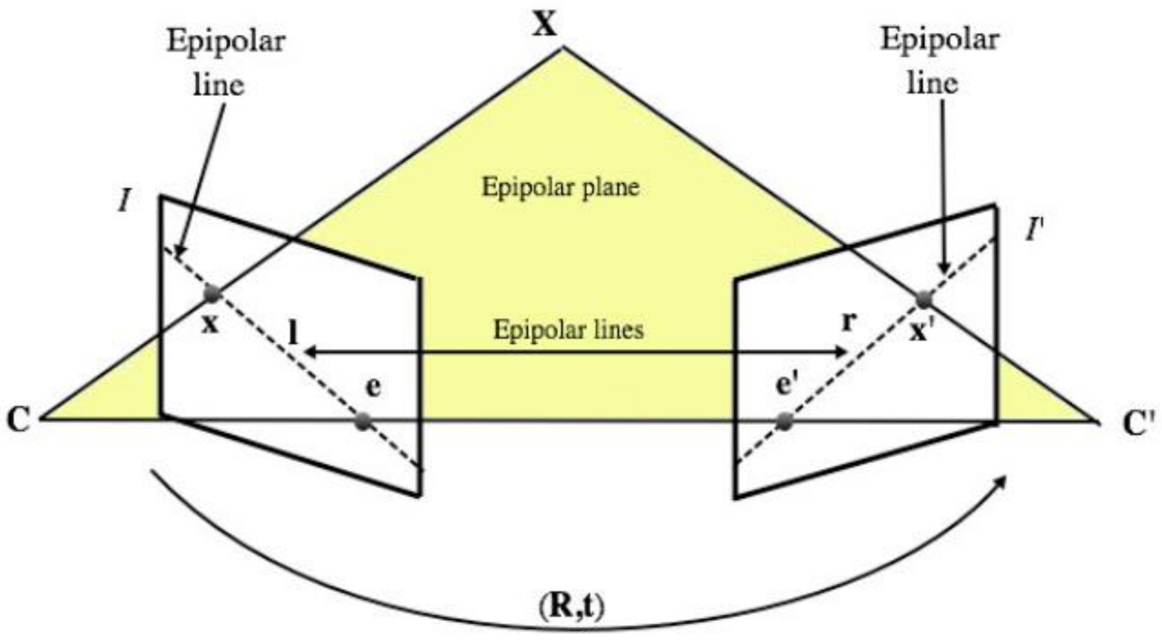


Let’s explain it in question 3.

Q3) Can you compute an accuracy for the projected points on im3 in Question 2? Comment on possible reasons for projection errors?

The occurrence of errors in the code can be attributed to errors in the code logic, sensitivity of operations or possible inaccuracies in manual selection of pixels.

Calculation of the basis matrix without taking into account errors in internal and external parameters can contribute to inaccuracies. Specifically, the calculated epipolar lines do not intersect perfectly because one passes from the bottom and the other from the top. We ignore this and assume that they are aligned lines. Ignoring these gaps during calculation will lead to errors. It is important to perform additional operations within the base matrix to improve accuracy. Ideally, using techniques such as least squares and normalization will yield the most accurate results. However, limitations in the available data may restrict the application of these advanced operations directly within the underlying matrix.



I want to thank my friend Nusret Taha Ege for helping me with second question code logic 😉