# 

**EE6222: MACHINE VISION**

**Project Report (Assignment 2)**

**Authors:**

Chin Zhi Wei (U1821267H)

Tan Chuan Xin (U1821755B)

Teo Chen Ning (U1820456K)

**Date of submission:**

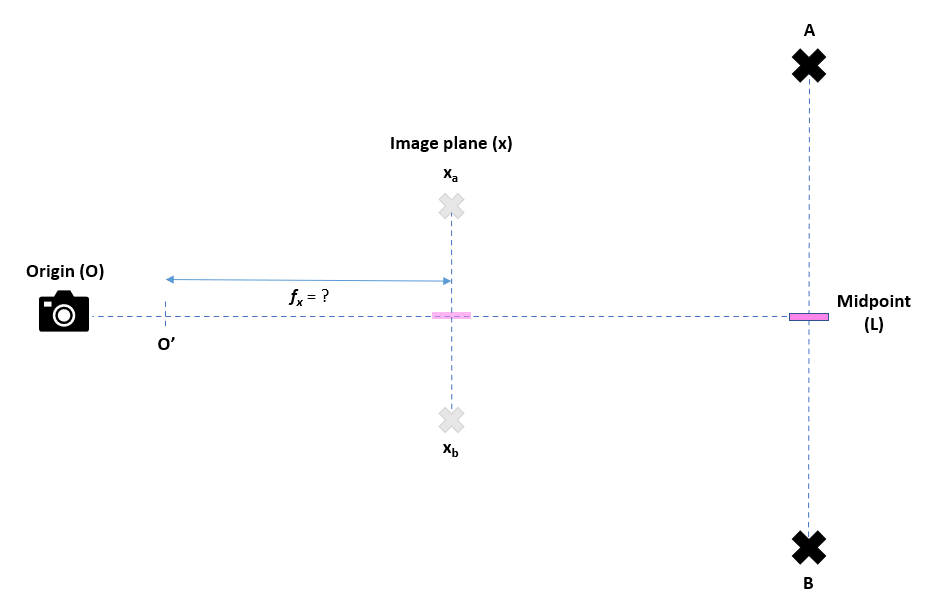
4 November 2021

# Question 1

**Find the focal length *f* of your hand phone (in pixels). You may use a real person or printed figure, and include one figure of the settings in your report. Make sure you turn the camera’s “zooming/auto-focusing” off. (40 marks)**

We used two crosses with a midpoint (in pink tape) as denoted below. Pictures were taken at a varying distance from the phone camera (**O**) to the physical midpoint of the two crosses, denoted by the distance **OL**.

The diagram below depicts how the experiment was set up. We vary the distance **OL** between 1240mm and 1800mm, and measure the pixel distance (**xaxb**) between the two crosses in the resulting image plane. We are trying to find the focal length ***fx*** in terms of its pixel value. The physical distance between the two crosses (**AB**) is fixed at 1200mm.



*Figure 1: Experimental setup*

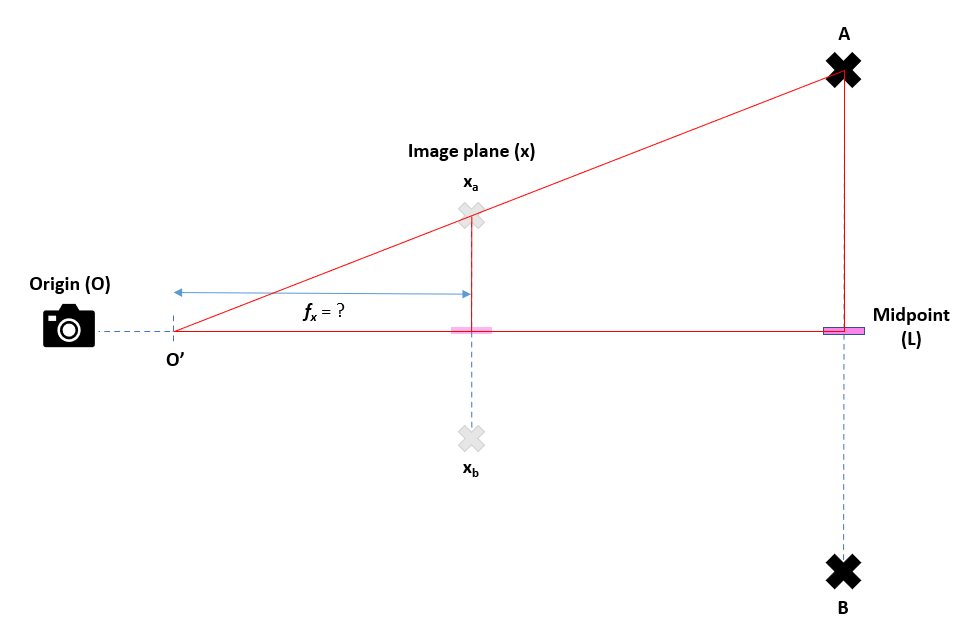
The settings used for the phone camera as well as the resulting images are listed below:

| **Phone model** | Samsung Galaxy S21+ |
| --- | --- |
| **Manufacturer’s focal length** | 26mm |
| **Zoom level** | 1.0x (no zoom) |
| **Focus mode** | Manual focus |
| **HDR mode** | Off |
| **Camera type** | Main (Not telephoto/selfie) |

| **Image 1**  **Taken at OL1 = 1240m** | **Image 2**  **Taken at OL2 = 1800mm** |
| --- | --- |

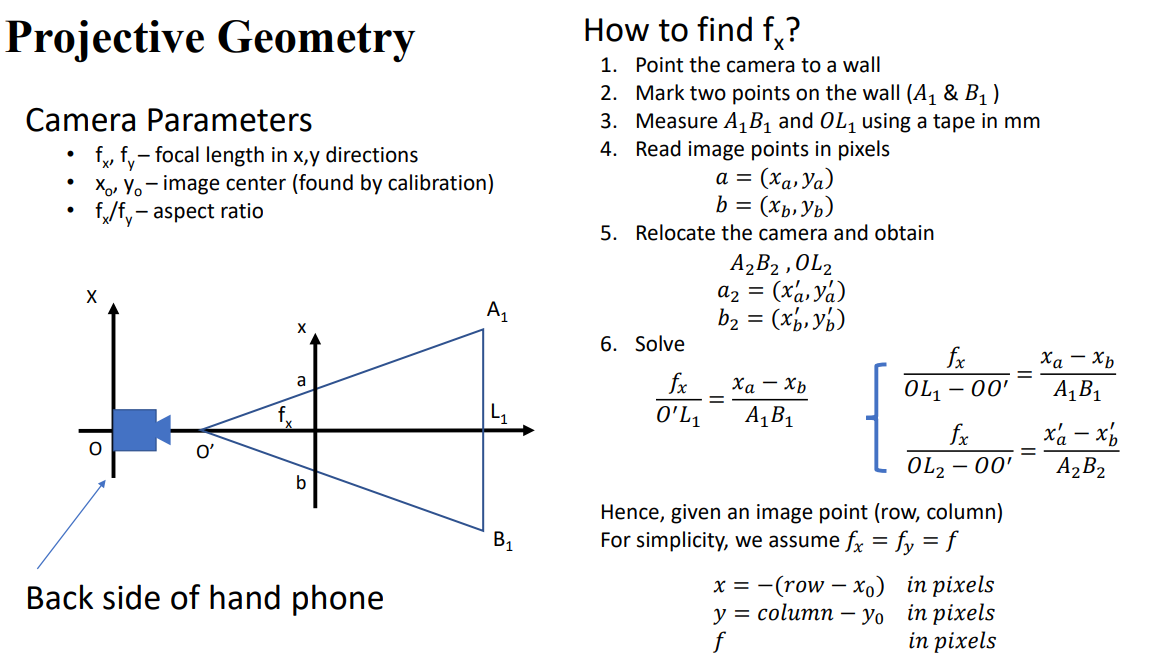
*Figure 2: Two images at different distances*

By projective geometry, we can see two similar triangles as shown below:



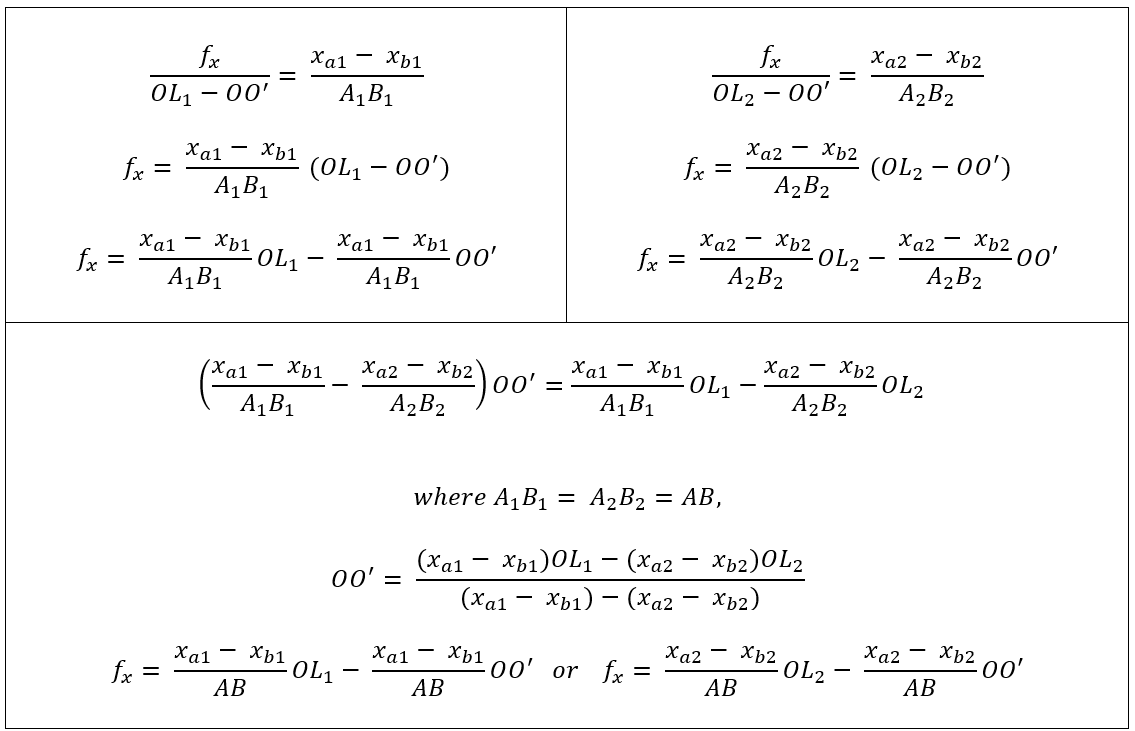
*Figure 3: Projective geometry*

We refer to Slide 4 of [2] 3D vision 1.pdf lecture slides for the equation to resolve both the pixel focal length ***fx***, and the actual focal length ***OO’***



*Figure 4: Projective Geometry equation from lecture slides*

Using two images taken at different distances ***OL****,*we are able to derive two sets of equations relating ***fx*** and ***OO’***, therefore we can solve for both variables.



*Figure 5: Formula derivation for* ***fx*** *and* ***OO’***

In our case, we have the following values after measurement from both images:

|  | **Image 1** | **Image 2** |
| --- | --- | --- |
| **OL** | 1240mm | 1800mm |
| **AB** | 1200mm | 1200mm |
| **x**a | 1111 | 925 |
| **xb** | 157 | 273 |

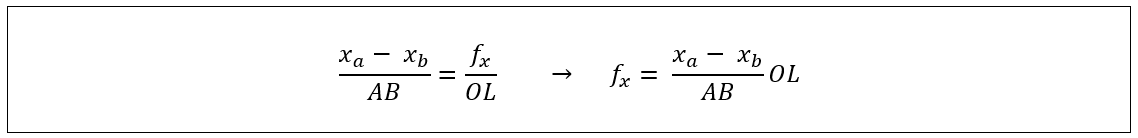
*Figure 6: Experimental values*

This allows us to solve the value of ***OO’*** and ***f*x** using the derivations in Figure 5



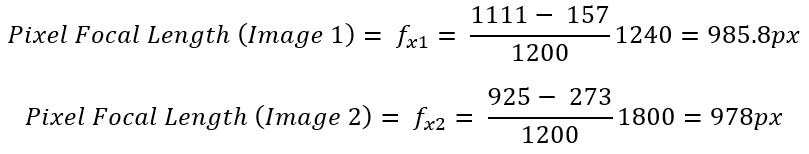
The focal length discovered is similar to the manufacturer’s specification of 26mm with around a 6mm difference.

Alternatively, we may make use of a simpler equation to determine the pixel focal length from only one image. This operates on the assumption that the distance ***OO’*** is negligible, that is, the distance of the lens to the focal point is negligible.



*Figure 7: Simplified formula with* ***OO’*** *being negligible*

Using the simplified equation, we can plug in the value of each image respectively, ending up with the following results:



These values are within +25px of the value found using two images. This suggests that it is possible to just approximate ***OO’*** as zero to simplify the calculations of the pixel focal length.

# Question 2

**Take two snaps of an outdoor scene, with a 5 to 10 degrees angle difference. You need to keep the angle as ground truth. (0 marks)**

| **0 degrees** | **9 degrees** |
| --- | --- |

*Figure 8: Two images at 9 degree angle difference*

# 

# Question 3

**Hand pick 8 points or more from one image, and find the matching points on the other image. These points should not be coplanar. You need to turn these points into n-vectors, and submit them into the equation for calculation. (10 marks)**

The 8 points selected in each image have been marked as red crosses with their corresponding pixel coordinates below:

| **0 degrees**  **Coords = [ (129, 322), (544, 1127),**  **(225, 133), (335, 168), (380, 341),**  **(210, 738), (210, 748), (224, 738)**  **]** | **9 degrees**  **Coords = [ (304, 318), (725, 1106),**  **(390, 114), (499, 137), (544, 309),**  **(385, 708), (386, 717), (399, 708)**  **]** |
| --- | --- |

*Figure 9: Choosing 8 random matching points in the two images*

# To turn these points into n-vectors, we will first homogenize these points. The pixel locations for each point in the image are a representation of a vector of shape (3,1) in world space (3D) which represents the x, y and z locations for each point. Since the points only exist as 2D vectors in the image, we will use the pixel focal length calculated in question 1 as the z-coordinate across all points to homogenize them.

We then adjust each point’s coordinates for the image center (i.e. make the image center the origin) to have a unified origin for our normalization process. One can make the center point of the image the origin by calculating the relative vectors from the center point as such:

Here, denotes the x and y coordinates of the center of the image. Now that we have an origin, we can normalize the vectors using their Euclidean norms.

% Focal length in pixels

F=961;

% Eight corresponding points of two images

pts1 = [[129 322]; [544 1127]; [225 133]; [335 168]; [380 341]; [210 738]; [210 748]; [224 738];];

pts2 = [[304 318]; [725 1106]; [390 114]; [499 137]; [544 309]; [385 708]; [386 717]; [399 708];];

% Center coordinates of image

x\_center = 960/2;

y\_center = 1280/2;

% Obtaining N-vectors

image1\_points = [pts1 ones(8,1)\*F];

image1\_points(: ,1) = image1\_points (: ,1) - X0 ;

image1\_points (: ,2) = image1\_points (: ,2) - Y0 ;

image1\_norm = vecnorm (image1\_points , 2 , 2) ;

image1\_points = image1\_points ./ image1\_norm ;

image2\_points = [pts2 ones(8,1)\*F];

image2\_points (: ,1) = image2\_points (: ,1) - X0 ;

image2\_points (: ,2) = image2\_points (: ,2) - Y0 ;

image2\_norm = vecnorm (image2\_points , 2 , 2) ;

image2\_points = image2\_points ./ image2\_norm ;

*Figure 10: Obtaining N-vectors of the points*

The final n-vectors for each image are as follows:

| image1\_points =  -0.3276 -0.2968 0.8970  0.0593 0.4512 0.8904  -0.2285 -0.4543 0.8611  -0.1342 -0.4369 0.8895  -0.0989 -0.2956 0.9502  -0.2692 0.0977 0.9581  -0.2689 0.1076 0.9571  -0.2562 0.0981 0.9616 | image2\_points =  -0.1711 -0.3130 0.9342  0.2236 0.4253 0.8770  -0.0819 -0.4785 0.8743  0.0175 -0.4637 0.8858  0.0628 -0.3250 0.9436  -0.0981 0.0702 0.9927  -0.0970 0.0795 0.9921  -0.0838 0.0703 0.9940 |
| --- | --- |

# Question 4

**Calculate the rotation angle from the matched points using the quaternion approach (pp 14 in [4]), or the SVD(in [3]).** **(40 marks)**

We are opting for the SVD approach in [3].

A 3x3 matrix W is obtained where∈ 𝑅 *3x3*. and are the N-vectors relative to the center of the images respectively, corresponding to *image1\_points* and *image2\_points* respectively.

% Creating W matrix

weight\_matrix = zeros (3 ,3) ;

for i = 1:8

temp = transpose ( image1\_points (i ,:) ) \* image2\_points (i ,:) ;

weight\_matrix = weight\_matrix + temp ;

end

*Figure 11: Computing the W matrix*

W can be further factorized into 3 matrices via singular value decomposition. Any invertible linear transformation can be broken down into a rotation/reflection, scaling and another rotation/reflection. As such, the rotational matrix R of the linear transformation can be computed via the following formula:

% SVD approach

[U , S , V ] = svd( W ) ;

rotational\_matrix = U \* transpose ( V ) ;

*Figure 12: Performing SVD of the W matrix to compute the Rotational matrix*

Using the rotational matrix R, we can calculate the rotation angle Ω as follows:

This gives us a rotation angle ꭥ of 10.2375°, which is close to the ground truth of 9°.

angleInRadians = acos((trace(rotational\_matrix)-1)/2);

angleInDeg = rad2deg(angleInRadians);

% angleInDeg = 10.2375 degrees;

*Figure 13: Obtaining rotational angle in degrees*

We can further double check the validity of our rotational matrix by calculating the N-vector of the rotational axis *l*. We obtain a N-vector that is approximately [0, -1, 0], indicating that we only have rotation about the y axis. This is accurate as we only rotated the phone 9° around a single axis.

l = transpose ([ rotational\_matrix(3 ,2)-rotational\_matrix(2 ,3) rotational\_matrix(1 ,3)-rotational\_matrix(3 ,1) rotational\_matrix(2 ,1)-rotational\_matrix(1 ,2)]) ;

l = l ./ vecnorm(l);

% N vector of rotational axis, [-0.1640 -0.9831 0.0814 ]

*Figure 14: Obtaining the rotational axis*