Homework 3 Questions

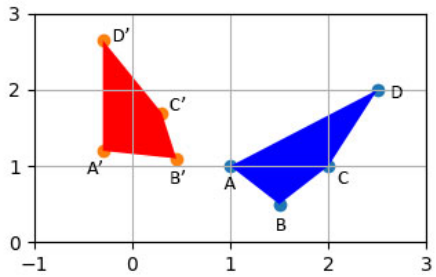
Document Instructions

- 5 questions **[10 + 8 + 4 + 13 + 3 = 40 points + 2 bonus points]**.
- Fill all your answers within the answer boxes, and **please do NOT remove the answer box outlines**.
- Questions are highlighted in the **orange boxes**, bonus questions are highlighted in **blue boxes**, answers should be recorded in the **green boxes**.
- Include code, images, and equations where appropriate.
- To identify all places where your responses are expected, search for 'TODO'.
- The answer box sizes have been set by the staff beforehand and will truncate your text if it goes beyond the limit. Please make sure your responses fit in the appropriate spaces. **Extra pages are not permitted unless otherwise specified.**
- Make sure your submission has the right number of pages to validate page alignment sanity (check the footer).
- Please make this document anonymous.

Gradescope Instructions

- When you are finished, compile this document to a PDF and submit it directly to Gradescope.
- The pages will be automatically assigned to the right questions on Gradescope *assuming you do not add any unnecessary pages*. **Inconsistently assigned pages will lead to a deduction of 2 points per misaligned page (capped at a maximum 6 point deduction).**

Q1: **[10 points]** Suppose we have a quadrilateral $ABCD$ and a transformed version $A'B'C'D'$ as seen in the image below.



$$\begin{aligned} A &= (1, 1) & A' &= (-0.3, 1.3) \\ B &= (1.5, 0.5) & B' &= (0.5, 1.1) \\ C &= (2, 1) & C' &= (0.3, 1.8) \\ D &= (2.5, 2) & D' &= (-0.3, 2.6) \end{aligned} \tag{1}$$

Let's assume that each point in $ABCD$ was approximately mapped to its corresponding point in $A'B'C'D'$ by a 2×2 transformation matrix $\mathcal{M}$.

e.g. if $X = \begin{pmatrix} x \\ y \end{pmatrix}$ and $X' = \begin{pmatrix} x' \\ y' \end{pmatrix}$, and $\mathcal{M} = \begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix}$

then $\begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \approx \begin{pmatrix} x' \\ y' \end{pmatrix}$

We would like to approximate $\mathcal{M}$ using least squares for linear regression.

(a) **[1 point]**

Rewrite the equation $\mathcal{M} \times X \approx X'$ into a pair of linear equations by expanding the matrix multiplication.

TODO: Replace each of the '_' below with x, y, x', y', or 0.

$$\begin{cases} x * m_{1,1} + y * m_{1,2} + 0 * m_{2,1} + 0 * m_{2,2} = x' \\ 0 * m_{1,1} + 0 * m_{1,2} + x * m_{2,1} + y * m_{2,2} = y' \end{cases}$$

(b) **[2 points]** With the quadrilaterals in question, there are 4 points that transform so we should expect to see 8 such equations (2 for each point) that use the transformation equation $\mathcal{M}$.

From these pairs of equations for each x-x' correspondence, we can construct a matrix $\mathcal{Q}$ and column vector b that satisfy

$$\mathcal{Q} \times \begin{pmatrix} m_{1,1} \\ m_{1,2} \\ m_{2,1} \\ m_{2,2} \end{pmatrix} = b$$

Note: Systems of linear equations are typically written in the form $\mathcal{A} \times x = b$, but since we have already defined $\mathcal{A}$ and x, we're writing it as $\mathcal{Q} \times m = b$

Find $\mathcal{Q}$ and b:

Replace each of the '__' below with a 0 or a coordinate value from $ABCD$ and $A'B'C'D'$.

TODO: your answer for (b) here

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1.5 & 0.5 & 0 & 0 \\ 0 & 0 & 1.5 & 0.5 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 2.5 & 2 & 0 & 0 \\ 0 & 0 & 2.5 & 2 \end{pmatrix} \times \begin{pmatrix} m_{1,1} \\ m_{1,2} \\ m_{2,1} \\ m_{2,2} \end{pmatrix} = \begin{pmatrix} -0.3 \\ 1.3 \\ 0.5 \\ 1.1 \\ 0.3 \\ 1.8 \\ -0.3 \\ 2.6 \end{pmatrix}$$

(c) **[1 point]** Our problem is now over-constrained, so we want to find values for $m_{i,j}$ that minimize the squared error between the approximated values for X' and the real X' values, i.e., we want to minimize $||\mathcal{Q} \times m - b||$.

To do this we use singular value decomposition to find the pseudoinverse of $\mathcal{Q}$, written as $\mathcal{Q}^\dagger$. We then multiply it by both sides, giving us:

$$\begin{aligned} \mathcal{Q}^\dagger \mathcal{Q} m &= \mathcal{Q}^\dagger b \\ m &\approx \mathcal{Q}^\dagger b. \end{aligned}$$

Thankfully, the computer can do all of this for us! `numpy.linalg.lstsq()` takes in our $\mathcal{Q}$ matrix and b vector, and returns approximations for m. Plug the values you wrote in part (c) into that function and write the returned $\mathcal{M}$ matrix here.

Note: You may need to reshape your output from `linalg.lstsq` to get the right dimensions.

Replace each of the '__' below with the value of $m_{i,j}$:

TODO: your answer for (c) here

$$M = \begin{pmatrix} m_{1,1} & m_{1,2} \\ m_{2,1} & m_{2,2} \end{pmatrix} = \begin{pmatrix} 0.632 & -0.9408 \\ 0.528 & 0.6768 \end{pmatrix}$$

(d) **[3 x 2 points]** Note that the least squares regression function that we use here is an approximation function, meaning that the values we derived for the transformation matrix are not exact. The same is true of our 8 point algorithm method of calculating the fundamental matrix.

Suppose there is some CV algorithm that uses an approximation such as the ones we learned about. Evaluate from the perspective of three stakeholders of your choice (e.g. the developer of the technology, the CEO of the company that develops the technology, a third party who uses your CV algorithm) how important transparency is surrounding margins of error. **[4-6 sentences]**

For the developer of the technology, the less margins of error I have, the better reputation I get. So I would say it's critical for developers. As for CEO, they would not know what margins of error the technology have unless developers tell CEO a true error. So it's also critical for them to know exact margions of error. As for a third party, if one doesn't know the exact margins of error, one could 100% believe the technology and rely on it, which could result in, for example, car accident.

Q2: [8 points] In lecture, you've learned that cameras can be represented by intrinsic and extrinsic matrices. These matrices can be used to calculate the projections of points within a 3D world onto 2D image planes. For this, we use *homogeneous coordinates*. The final 3×4 matrix is known as the *camera matrix*.

Recall that the transformation can be represented by the following expression:

$$\begin{pmatrix} f_x & s & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = w \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

where f is the focal point, r is the rotation matrix, t is the translation vector, w is some weighing/scaling factor, and (u, v) is the position of the point in the real world (x, y, z) projected on the 2D plane.

(a) **[2 points]** For each of the following, you are given the camera specifications and a sample 3D point from the real world.

> Fill in the camera's intrinsic and extrinsic matrices; then, perform the multiplications and perspective division (unhomogenize) to find the 2D coordinate of the projected point on the image.

(i) A camera with a focal length of 1 in both the x and y directions, a translation of 5 along the x-axis, and no skew or rotation.

TODO: Fill in the __ entries

$$M_{\text{intrinsic}} \times M_{\text{extrinsic}} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 30 \\ \text{-20} \\ 10 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 35 \\ -20 \\ 10 \end{pmatrix}$$

$$= 10 \times \begin{pmatrix} 3.5 \\ -2 \\ 1 \end{pmatrix}$$

(ii) A camera with focal length of 2 in both the x and y directions, a translation of 5 along the x-axis, and no skew or rotation.

TODO: Fill in the __ entries

$$= \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 30 \\ -20 \\ 10 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 70 \\ -40 \\ 10 \end{pmatrix}$$

$$= 10 \times \begin{pmatrix} 7 \\ -4 \\ 1 \end{pmatrix}$$

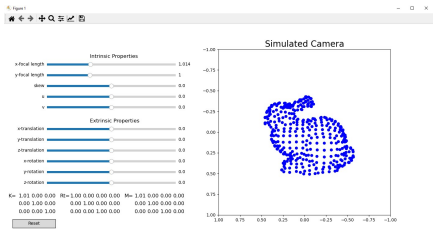
(b) **[2 points]**

Compare the two image coordinates you've calculated in parts a and b. Explain how each parameter affects the final image coordinate. **[2-3 sentences]**

Focal lengths behave like scaling factor. When focal length is two fold, the coordinate on image doubled.

(c) **[1 + 3 points]** In the questions folder, we've provided stencil code for a camera simulation in `camera_simulation.py`. Given a camera matrix, the simulator visualizes an image that a camera would produce.

Please implement `calculate_camera_matrix()` by calculating the camera matrix using the parameters given in the code (see stencil for more detail). When successful, you will see a bunny rendered as dots (see below). Paste your code for this function and attach a screenshot of the working demo once you finish. Play around with the sliders to see how different parameters affect the projection!




```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib.widgets import Slider, Button

# Initial random matrices
initial_intrinsic_matrix_to_replace = np.random.rand(3,3)
initial_extrinsic_matrix_to_replace = np.random.rand(3,4)
initial_camera_matrix_to_replace = np.random.rand(3,4)

# Setting up the point cloud
file_data_path= "./bunny.xyz"
point_cloud = np.loadtxt(file_data_path, skiprows=0,
                                 max_rows=1000000)
# center it
point_cloud -= np.mean(point_cloud,axis=0)
# homogenize
point_cloud = np.concatenate((point_cloud, np.ones((
                                 point_cloud.shape[0], 1))),
                                 axis=1)
# move it in front of the camera
point_cloud += np.array([0,0,-0.15,0])

def calculate_camera_matrix(tx, ty, tz, alpha, beta, gamma,
                                  fx, fy, skew, u, v):
    """
    This function should calculate the camera matrix using
                                      the given
    intrinsic and extrinsic camera parameters.
    We recommend starting with calculating the intrinsic
                                      matrix (refer to lecture
```

```
                                    8).
    Then calculate the rotational 3x3 matrix by calculating
                                     each axis separately and
    multiply them together.
    Finally multiply the intrinsic and extrinsic matrices
                                    to obtain the camera
                                    matrix.
    :params tx, ty, tz: Camera translation from origin
    :param alpha, beta, gamma: rotation about the x, y, and
                                     z axes respectively
    :param fx, fy: focal length of camera
    :param skew: camera's skew
    :param u, v: image center coordinates
    :return: [3 x 4] NumPy array of the camera matrix, [3 x
                                     4] NumPy array of the
                                    intrinsic matrix, [3 x 4]
                                     NumPy array of the
                                    extrinsic matrix
    """
    ########################
   intrinsic_matrix = np.array([fx, skew, 0, 0, fy, 0, 0, 0
                                   , 1]).reshape(3,3)
    rotate_x = np.array([1,0,0,0, np.cos(alpha), -np.sin(
                                    alpha), 0, np.sin(alpha),
                                     np.cos(alpha)]).reshape(
                                    3,3)
    rotate_y = np.array([np.cos(beta), 0, np.sin(beta), 0,
                                    1, 0, -np.sin(beta), 0,
                                    np.cos(beta)]).reshape(3,
                                    3)
    rotate_z = np.array([np.cos(gamma), -np.sin(gamma), 0,
                                    np.sin(gamma), np.cos(
                                    gamma), 0, 0, 0, 1]).
                                    reshape(3,3)
    extrinsic_matrix_without_traslation = rotate_z @
                                    rotate_y @ rotate_x

    extrinsic_matrix = np.concatenate((
                                    extrinsic_matrix_without_traslation
                                    , np.array([tx,ty,tz]).
                                    reshape(3,1)), axis=1)
    print(extrinsic_matrix.shape)
    camera_matrix = intrinsic_matrix @ extrinsic_matrix
    ########################
    return (initial_camera_matrix_to_replace,
    initial_intrinsic_matrix_to_replace,
    initial_extrinsic_matrix_to_replace)

def find_coords(camera_matrix):
    """
    This function calculates the coordinates given the
                                    student's calculated
                                    camera matrix.

    Normalizes the coordinates.
    Already implemented.
    """
```

```
    coords = np.matmul(camera_matrix, point_cloud.T)
    return coords / coords[2]









    ################################################
    # YOU MAY USE THIS ADDITIONAL PAGE

    # WARNING: IF YOU DON'T END UP USING THIS PAGE
    # KEEP THESE COMMENTS TO MAINTAIN PAGE ALIGNMENT
    ################################################
```

Q3: **[4 points]** Given a stereo pair of cameras:

(a) **[2 points]**

Briefly describe triangulation (using an image might be helpful) **[3-4 sentences]**

Given points' projections onto two, or more, images, we want to determine the points in 3D. This is called triangulation. We uses a project line of a 3D onto 2D to generate another project lines for each possible point on a first line or vice versa.

(b) **[2 points]**

Why is it not possible to find an absolute depth for each point when we don't have calibration information for our cameras? Note that absolute depth refers to depth with respect to the camera as opposed to relative depth, which is with respect to another object in the scene. **[3 - 4 sentences]**

To solve for disparity of an image, we need Z, f, T, etc which are all calibration information. Without theses leads, we can't solve for disparity, which is related to absolute depth of each point in 3d. We can only generate image that doesn't have a sense of distance.

Q4: **[13 points]** Given the algorithms that we've learned in computer vision, we know that whether we can find/calculate the essential matrix, the fundamental matrix, or both depends on the setup of the cameras and images. You are given three datasets of an object of unknown geometry:

(i) A video circling the object;

(ii) A stereo pair of calibrated cameras capturing two images of the object; and

(iii) Two images we take of the object at two different camera poses (position and orientation) using the same camera but with different lens zoom settings.

(a) **[3 × 1 points]**

For each of the above setups, what calculations can we perform?

(i) Setup 1

TODO: Select the right option:

Essential Matrix	☐
Fundamental Matrix	■
Both	☐

(ii) Setup 2

TODO: Select the right option:

Essential Matrix	☐
Fundamental Matrix	☐
Both	■

(iii) Setup 3

TODO: Select the right option:

Essential Matrix	☐
Fundamental Matrix	■
Both	☐

(b) **[3 × 1 points]**

State an advantage and disadvantage of using each setup for depth reconstruction **[2 - 3 sentences]**

(i) Setup 1

Video is continuous data of images. Also, we can get 360 degree view of an object. However, we don't know intrinsic matrix of a camera/recorder that took a video.

(ii) Setup 2

As we have a pair of calibrated cameras, we can get intrinsic matrix of each one, which gives us a essential Matrix (and also fundamental matrix as well). But depending on where we set two cameras, they might not be useful to get true depth of pixels of 3D.

(iii) Setup 3

This one is similar to the second case. But since this camera is not calibrated, we can't get essential matrix. But since we have various settings of a camera, we could get a little more better depth of 3D pixels.

(c) **[3 × 1 points]**

Name an application scenario for each of the different setups **[1 - 2 sentences]**

(i) Setup 1

Amazon household simulation. There's a service in Amazon app that when we upload video, it can detect depth of user's environment and show how house furniture fit in.

(ii) Setup 2

Autonomous driving uses this skill. Some companies might use Lidar but some companies use several cameras to detect environment.

(iii) Setup 3

We can use this when we want to detect objects that could be occluded. As we use different positions and zoom ratio, we can more correctly detect exact depth. This might be useful in navigating in mountain where there are many obstacles that prevent your sight.

(d) **[4 points]** The differences between the collection methods for these three datasets are crucial in terms of what calculations are possible - and therein which applications they are most useful in.

From a non-technical standpoint, can you think of a scenario why you may prefer one of these data collection setups to another? Why is it important to know what data collection methods have been used to build a particular dataset? **[5-7 sentences]**

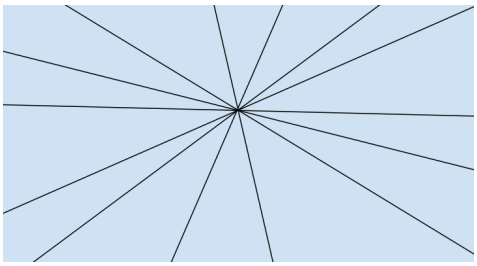
It's important to know what data collection methods we used because we want to know what conditions/restriction/parameters/etc are available or not when building a model or when extracting features out of data set. I would prefer using first method. I think more data can give use more accurate result. As a video is continuous sets of images, I can get much more data then a couple of images.

Q5: **[3 points]** In two-view camera geometry, what do the following epipolar lines say about the cameras' relative positions? **[1 - 2 sentences each]**

Tip: The Spring '22 course staff created an interactive demo to explore the different scenarios and get a better feel for epipolar geometry.

(a) **[1 point]**

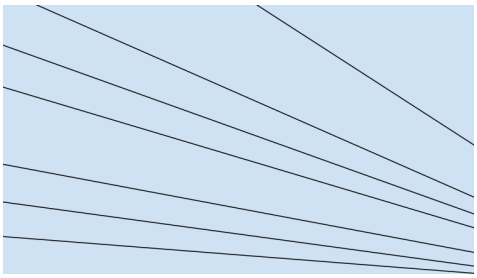
Radiate out of a point on the image plane.



Cameras are located at back and forth.

(b) **[1 point]**

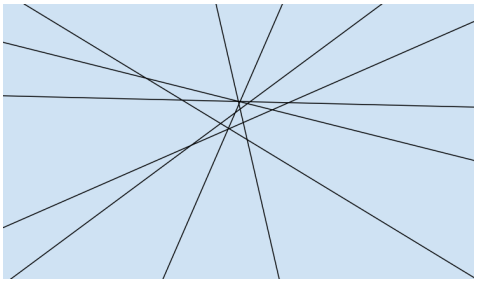
Converge to a point outside of the image plane.



A camera on the right is right at the edge of original camera's view plane.

(c) **[1 point]**

Notice the misalignment of the epipolar lines in the image below? What went wrong in the calculation of the fundamental matrix and how can we fix it?
Hint: Check slides from the lecture on stereo geometry.



F is not correct. Should fix F, a fundamental matrix.

Discussion Attendance:

Extra Credit: [2 points]

Please mark this box only if you've attended the discussion session in person.

■ I attended the discussion session on DATE

Feedback? (Optional)

Please help us make the course better. If you have any feedback for this assignment, we'd love to hear it!