CSE 152 : Introduction to Computer Vision, Spring 2018 – Assignment 2

Instructor: Ben Ochoa

Assignment Published On: Wednesday, April 11, 2018

Due On: Wednesday, April 25, 2018, 11:59 PM

Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains both math and programming problems.
- All solutions must be written in this notebook
- For the Math problems you may use Markdown/LATEX or you can work it out on paper and upload the scanned copy after merging with the .ipynb PDF. Remember to show work and describe your solution.
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. You must also submit this notebook as .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- It is highly recommended that you begin working on this assignment early.

Problem 1: Geometry [20 points]

Consider a line in the 2D plane, whose equation is given by $a\tilde{x} + b\tilde{y} + c = 0$. This can equivalently be written as $\boldsymbol{l}^\top \boldsymbol{x} = 0$, where $\boldsymbol{l} = (a, b, c)^\top$ and $\boldsymbol{x} = (\tilde{x}, \tilde{y}, 1)^\top$. Noticing that $\boldsymbol{x}$ is a homogeneous representation of $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y})^\top$, we can view $\boldsymbol{l}$ as a homogeneous representation of the line $a\tilde{x} + b\tilde{y} + c = 0$. We see that the line is also defined up to a scale since $(a, b, c)^\top$ and $k(a, b, c)^\top$ with $k \neq 0$ represents the same line. All points (x, y) that lie on the line $a\tilde{x} + b\tilde{y} + c = 0$ satisfy the equation $\boldsymbol{l}^\top \boldsymbol{x} = 0$.

> **Statement 1:** A point $\boldsymbol{x}$ lies on the line $\boldsymbol{l}$ if and only if $\boldsymbol{l}^\top \boldsymbol{x} = \boldsymbol{x}^\top \boldsymbol{l} = 0$

1. [4 points] Using Euclidean coordinates, find the equation of the line perpendicular to the family of lines $y = x + \lambda$ whereas $\lambda \in (-\infty, \infty)$ and at a distance d from the origin. Your answer should be represented only in terms of the given parameters.

2. [6 points] Prove the following two statements that follow from **Statement 1**.
   a). The cross product between two points gives us the line connecting the two points
   b). The cross product between two lines gives us their point of intersection

3. [4 points] What is the line, in homogenous coordinates, joining the inhomogeneous points $(1, 4)$ and $(4, 5)$.

4. [6 points] When a rectangle $ABCD$ is observed under pinhole perspective, the image will be arbitrary quadrilateral $A'B'C'D'$. Answer the following questions using your newly learned skilled in working with homogeneous representations.
   a). [3 points] For any arbitrarily imaged rectangle $ABCD$ with non zero area, can $A'B'C'D'$ ever be a non-convex quadrilateral? Explain the intuition behind your answer. (Note: A convex polygon is a simple polygon (not self-intersecting) in which no line segment between two points on the boundary ever goes outside the polygon.)
   b). [3 points] Let $A' = (t, t)$, $B' = (t, 6t)$, $C' = (4t, 6t)$ and $D' = (2t, 4t)$ be the vertices of the image. Find all the vanishing points of the quadrilateral (i.e. the points of intersections of pairs of opposite lines through $\{A'B', C'D'\}$ and $\{B'C', A'D'\}$ given $t = 1$.

Problem 2: Image Formation and Rigid Body Transformations [20 points]

In this problem we will practice rigid body transformations and image formations through the projective camera model. The goal will be to photograph the following four points $\boldsymbol{X}_1 = [\text{-}1\ \text{-}0.5\ 2]^T$, $\boldsymbol{X}_2 = [1\ \text{-}0.5\ 2]^T$, $\boldsymbol{X}_3 = [1\ 0.5\ 2]^T$, $\boldsymbol{X}_4 = [\text{-}1\ 0.5\ 2]^T$ in the world coordinate frame. First, recall the following formula for rigid body transformation

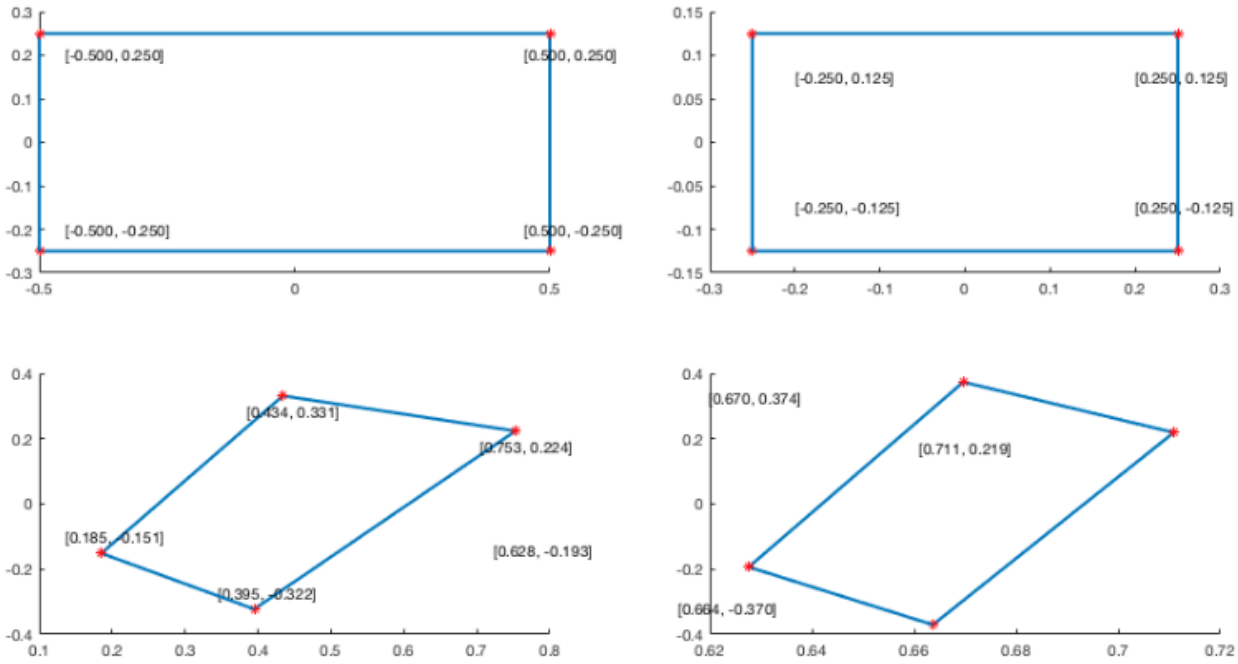
$$\widetilde{\boldsymbol{X}}_{cam} = R\widetilde{\boldsymbol{X}} + \boldsymbol{t}$$

Where $\widetilde{\boldsymbol{X}}_{cam}$ is the point coordinate in the camera coordinate system. $\widetilde{\boldsymbol{X}}$ is a point in the world coordinate frame, and R and $\boldsymbol{t}$ are the rotation and translation that transform points from the world coordinate frame to the camera coordinate frame. Together, R and $\boldsymbol{t}$ are the *extrinsic* camera parameters. Once transformed to the camera coordinate frame, the points can be photographed using the 3×3 camera calibration matrix K, which embodies the *intrinsic* camera parameters, and the canonical projection matrix $[\mathrm{I}|\mathbf{0}]$. Given K, R, and $\boldsymbol{t}$, the image of a point $\widetilde{\boldsymbol{X}}$ is $\boldsymbol{x} = \mathrm{K}[\mathrm{I}|\mathbf{0}]\boldsymbol{X}_{\mathrm{Cam}} = \mathrm{K}[\mathrm{R}|\boldsymbol{t}]\boldsymbol{X}$, where the homogeneous points $\boldsymbol{X}_{\mathrm{Cam}} = (\widetilde{\boldsymbol{X}}_{\mathrm{Cam}}^{\top}, 1)^{\top}$ and $\boldsymbol{X} = (\widetilde{\boldsymbol{X}}^{\top}, 1)^{\top}$. We will consider four different settings of focal length, viewing angles and camera positions below.

a). The extrinsic transformation matrix,

b). Intrinsic camera matrix under the perspective camera assumption.

c). Calculate the image of the four vertices and plot using the supplied **plot_points** function (see e.g. output in figure below).



1. [No rigid body transformation]. Focal length = 1. The optical axis of the camera is aligned with the z-axis.
2. [Translation]. $\boldsymbol{t} = [0\ 0\ 1]^T$. The optical axis of the camera is aligned with the z-axis.

3. [Translation and Rotation]. Focal length = 1. R encodes a 30 degrees around the z-axis and then 60 degrees around the y-axis. $\boldsymbol{t} = [0\ 0\ 1]^T$.
4. [Translation and Rotation, long distance]. Focal length = 5. R encodes a 30 degrees around the z-axis and then 60 degrees around the y-axis. $\boldsymbol{t} = [0\ 0\ 13]^T$.

We will not use a full intrinsic camera matrix (e.g. that maps centimeters to pixels, and defines the coordinates of the center of the image), but only parameterize this with f, the focal length. In other words: the only parameter in the intrinsic camera matrix under the perspective assumption is f.

For all the four cases, include a image like above. Note that the axis are the same for each row, to facilitate comparison between the two camera models. Note: the angles and offsets used to generate these plots may be different from those in the problem statement, it's just to illustrate how to report your results.

Also, Explain why you observe any distortions in the projection, if any, under this model.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# convert points from euclidian to homogeneous
def to_homog(points):
    """
    your code here
    """
    return points_homog

# convert points from homogeneous to euclidian
def from_homog(points_homog):
    """
    your code here
    """
    return points

# project 3D euclidian points to 2D euclidian
def project_points(P_int, P_ext, pts):
    """
    your code here
    """
    #return the 2d euclidean points
    pts_2d=np.zeros([2,1])
    return pts_2d

def camera1():
    """
    replace with your code
    """
    P_int_proj = np.eye(3,4)
    P_ext = np.eye(4,4)
    return P_int_proj, P_ext

def camera2():
    """
    replace with your code
    """
    P_int_proj = np.eye(3,4)
    P_ext = np.eye(4,4)
    return P_int_proj, P_ext

def camera3():
    """
    replace with your code
    """
    P_int_proj = np.eye(3,4)
    P_ext = np.eye(4,4)
    return P_int_proj, P_ext

def camera4():
    """
    replace with your code
    """
    P_int_proj = np.eye(3,4)
    P_ext = np.eye(4,4)
```

```
    return P_int_proj, P_ext


#########################################################
# test code. Do not modify
#########################################################

def plot_points(points, title='', style='.-r', axis=[]):
    inds = list(range(points.shape[1]))+[0]
    plt.plot(points[0,inds], points[1,inds],style)
    if title:
        plt.title(title)
    if axis:
        plt.axis('scaled')
        #plt.axis(axis)

def main():
    point1 = np.array([[-1,-.5,2]]).T
    point2 = np.array([[1,-.5,2]]).T
    point3 = np.array([[1,.5,2]]).T
    point4 = np.array([[-1,.5,2]]).T
    points = np.hstack((point1,point2,point3,point4))

    for i, camera in enumerate([camera1, camera2, camera3, camera4]):
        P_int_proj, P_ext = camera()
        plt.subplot(1, 2, 1)
        plot_points(project_points(P_int_proj, P_ext, points), title='Ca
mera %d Projective'%(i+1), axis=[-.6,.6,-.6,.6])
        plt.show()

main()
```

Problem 3: Image Rendering [20 points]

In this exercise, we will render the image of a face with two different point light sources using a Lambertian reflectance model. We will use two albedo maps, one uniform and one that is more realistic. The face heightmap, the light sources, and the two albedo are given in facedata.npy for Python (each row of the `lightsource' variable encode a light location). The data from facedata.npy is already provided to you.

Note: Please make good use out of subplot to display related image next to eachother.

3.1 Plot the face in 2-D [2 pts]

Plot both albedo maps using imshow. Explain what you see.

3.2 Plot the face in 3-D [2 pts]

Using both the heightmap and the albedo, plot the face using plot_surface. Do this for both albedos. Explain what you see.

3.3 Surface normals [8 pts]

Calculate the surface normals and display them as a quiver plot using quiver in matplotlib.pyplot in Python. Recall that the surface normals are given by

$$[-\frac{\delta f}{\delta x}, -\frac{\delta f}{\delta y}, 1].$$

Also, recall, that each normal vector should be normalized to unit length.

3.4 Render images [8 pts]

For each of the two albedos, render three images. One for each of the two light sources, and one for both light-sources combined. Display these in a 2×3 subplot figure with titles. Recall that the general image formation equation is given by

$$I = a(x, y)\hat{\boldsymbol{n}}(x, y)^{\top}\hat{\boldsymbol{s}}(x, y)\frac{s_0}{(d(x, y))^2}$$

where $a(x, y)$ is the albedo for pixel (x, y), $\hat{n}(x, y)$ is the corresponding surface normal, $\hat{s}(x, y)$ the light source direction, s_0 the light source intensity, $d(x, y)$ the distance to the light. Let the light source intensity be 1 and do *not* make the `distant light source assumption'. Use imshow with appropriate keyword arguments .

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
import matplotlib.patches as patches
# Load facedata.npy as ndarray
face_data = np.load('facedata.npy',encoding='latin1')
# Load albedo matrix
albedo = face_data.item().get('albedo')
# Load uniform albedo matrix
uniform_albedo = face_data.item().get('uniform_albedo')
# Load heightmap
heightmap = face_data.item().get('heightmap')
# Load light source
light_source = face_data.item().get('lightsource')
```