COMPSCI 527 Homework 7

Problem 0 (3 points)

Group Members: Beck Addison, Shivam Kaul, Gin Wang

Part 1: Vector Geometry

Problem 1.1 (Exam Style)

A non-square matrix can have either a left inverse or a right inverse depending on whether it exhibits a full column rank or a full row rank respectively. For a matrix with the shape of $3X2$, the number of columns is lesser than the number of rows and it is hence a full column rank type matrix.

Therefore, only a left inverse exists for such matrices.

Problem 1.2 (Exam Style)

Let A be the square matrix with the left inverse L.

$$LA = I$$
$$L = A^{-1}$$
$$AL = I$$

This implies that the left inverse of A is also the right inverse of A.

Therefore, $L = R$.

Problem 1.3 (Exam Style)

Let **c** be a unit vector in the hyperplane orthogonal to the unit vector **n**. The projection of the vector **a** onto **c** is given by:

$$h = cc^T a$$

We also know that,

$$p = nn^T a$$

This implies

$$h + p = cc^T a + nn^T a = a$$
$$(cc^T + nn^T)a = Ia$$

Therefore,

$$H = cc^T = I - nn^T$$

Problem 1.4 (Exam Style)

The unit vector **c** can be obtained by taking the cross-product of the vectors **a** and **b** and then dividing that vector by its norm.

$$c = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

$$c = \begin{vmatrix} 0 & 2 & 0 \\ 1 & 3^{1/2} & 0 \end{vmatrix} = [0, 0, -2]$$

Therefore, the required unit vector is $[0, 0, -1]$

Problem 1.5 (Exam Style)

The volume of the parallelepiped thus formed is given by the magnitude triple-product of the three vectors **a**, **b** and **c**.

$$c = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix} = \begin{vmatrix} 1 & 2 & 0 \\ -1 & 3 & 1 \\ 0 & 0 & -4 \end{vmatrix} = -20$$

Therefore, the volume of the parallelepiped is **20**.

Problem 1.6 (Exam Style)

The condition for a face of a cube in 3D space to be visible to a pinhole camera at position $\mathbf{c}$ with a center at $\mathbf{f}$ and a face orthogonal to the outer matrix $\mathbf{n}$ is given by the following equation:

$$\mathbf{n^T}(\mathbf{f}-\mathbf{c}) < 0$$

To understand this condition, we should first consider the subtraction of $\mathbf{f}-\mathbf{c}$; this generates a vector that is effectively the "angle of attack" of the camera from its position relative to one of the faces of the cube. Then we dot this with $\mathbf{n}$ - this dot product describes the angle between the camera's perspective and the surface of the face of interest. If this value is less than zero (i.e. the angle is greater than 90 degrees), then we can say that the face of the cube is visible. A unique situation is when the face of the cube and the camera perspective are oriented at a 90 degree angle - if we consider this to be a visible case (I suppose the edge is visible, but not the face), then the equation becomes ≤ 0 rather than < 0.

Part 2: Rotations in the Plane

Problem 2.1 (Exam Style)

Consider an object reference frame rotated counter-clockwise by an angle θ with respect to a world reference frame in 2D. Let there be a point P whose coordinates with respect to the world reference frame is (x, y) and with respect to the object reference frame be (x_{obj}, y_{obj}). Our objective is to determine a rotation matrix R that transforms (x, y) to (x_{obj}, y_{obj}).

Using the concept of similarity of triangles, we obtain the following relations:

$$x_{obj} = xcos\theta + ysin\theta$$
$$y_{obj} = -xsin\theta + ycos\theta$$

Therefore, the rotation matrix R is given by:

$$\begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}$$

Problem 2.2 (Exam Style)

When we consider the rotation of a point by an angle θ in the counter clockwise direction with respect to the world reference system, that is equivalent to the rotation of the object reference system by an angle of $-\theta$ with respect to the world system.

Therefore, the rotation matrix in this case $\tilde{R}$ is given by:

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Problem 2.3 (Exam Style)

Consider an object reference frame be rotated first by an angle θ_1 and then further by an angle θ_2 in the counter-clockwise direction with respect to the world reference frame.

The final coordinates of a point P whose coordinates with respect to the world frame are (x, y) after the transformations is given by: $$ [x*{final},y*{final}] =

$$\begin{bmatrix} cos\theta_2 & sin\theta_2 \\ -sin\theta_2 & cos\theta_2 \end{bmatrix}$$
$$\begin{bmatrix} cos\theta_1 & sin\theta_1 \\ -sin\theta_1 & cos\theta_1 \end{bmatrix}$$

[x,y]

\ =

$$\begin{bmatrix} cos(\theta_1+\theta_2) & sin(\theta_1+\theta_2) \\ -sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) \end{bmatrix}$$

[x,y] $$

Let us now consider the case in which we first rotate the object frame by an angle θ_2 in the counter-clockwise direction with respect to the world reference frame and the further by θ_1. In this scenario, the final coordinates of the point P are obtained as follows:

$$\begin{aligned} [x_{final'}, y_{final'}] &= \begin{bmatrix} cos\theta_1 & sin\theta_1 \\ -sin\theta_1 & cos\theta_1 \end{bmatrix} \begin{bmatrix} cos\theta_2 & sin\theta_2 \\ -sin\theta_2 & cos\theta_2 \end{bmatrix} [x, y] \\ &= \begin{bmatrix} cos(\theta_1+\theta_2) & sin(\theta_1+\theta_2) \\ -sin(\theta_1+\theta_2) & cos(\theta_1+\theta_2) \end{bmatrix} [x, y] \end{aligned}$$

Clearly, we can see that:

$$[x_{final}, y_{final}] = [x^{'}_{final}, y^{'}_{final}]$$

Hence, the rotations $R_1 R_2$ is the same as $R_2 R_1$ and this is the commutative property that they satisfy.

Part 3: Euler's Rotation Theorem

Problem 3.1 (Exam Style)

"For every rotation there is a single line in space whose points do not move"

From the matrix obtained in 2.1, we have

$$R = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}$$

The eigenvalues of R are as follows:

$$(cos(\theta) - \lambda)^2 + sin^2(\theta) = 0$$
$$cos^2(\theta) + \lambda^2 - 2\lambda cos(\theta) + sin^2(\theta) = 0$$
$$\lambda^2 - 2cos(\theta)\lambda + 1 = 0$$
$$\lambda = cos(\theta) \pm i(sin(\theta))$$

The corresponding eigenvectors are, for $\lambda = cos(\theta) \pm i(sin(\theta))$:

$$\begin{bmatrix} -isin(\theta) & sin(\theta) \\ -sin(\theta) & -isin(\theta) \end{bmatrix} \mathbf{v} = 0$$

Therefore the eigenvectors are:

$$\mathbf{v_1} = \begin{bmatrix} 1 \\ i \end{bmatrix}, \quad \mathbf{v_2} = \begin{bmatrix} 1 \\ -i \end{bmatrix}$$

The fact that the eigenvalues and eigenvectors are complex and yet unit length suggests that no scaling occurs in the plane and that the axis of rotation has no components in that plane either. Thus, a rotation happens about an axis orthogonal to the plane.

Problem 3.2

In [204]:
```
import numpy as np

def rotation_basis(r):
    assert r.shape == (3,), 'a must be a 3-vector'
    u, _, vt = np.linalg.svd(np.expand_dims(r, axis=1))
    if vt[0, 0] < 0:  # make sure that a is not flipped
        u = -u
    if np.linalg.det(u) < 0:  # make sure that u is right-handed
        u[:, (1, 2)] = u[:, (2, 1)]
    return u
```

In [205]:
```python
import urllib.request
import pickle
from os import path as osp


def retrieve(file_name, semester='spring21', homework=7):
    if osp.exists(file_name):
        print('Using previously downloaded file {}'.format(file_name))
    else:
        fmt = 'https://www2.cs.duke.edu/courses/{}/compsci527/homework/{}/{}'
        url = fmt.format(semester, homework, file_name)
        urllib.request.urlretrieve(url, file_name)
        print('Downloaded file {}'.format(file_name))

rotations_pickle = 'rotations.pkl'
retrieve(rotations_pickle)
with open(rotations_pickle, 'rb') as file:
    rotations = pickle.load(file)
```

```
Using previously downloaded file rotations.pkl
```

In [206]:
```python
def print_array(a, name=None, unit=None, indent=0):
    dims = a.ndim
    assert dims == 1 or dims == 2, 'Can only print vectors or matrices'
    tabs = '\t' * indent
    prefix = tabs if name is None else '{}{}: '.format(tabs, name)
    suffix = '' if unit is None else ' {}'.format(unit)
    with np.printoptions(precision=3, suppress=True):
        if dims == 1:
            print('{}{}{}'.format(prefix, a, suffix))
        else:
            if len(suffix):
                suffix = ' ({})'.format(suffix)
            print('{}'.format(prefix, suffix))
            for row in a:
                print(tabs, '\t', row, sep='', end='\n')
```

In [207]:
```python
def rotation_matrix_to_vector(R):

    eigens = np.linalg.eig(R)[1][:,-1]
    a = np.real(eigens)

    b, c = rotation_basis(a)[:,(1,2)].T # get b and c
    s = R @ b # get s as a rotation of b

    angle_of_rotation = np.arctan2(np.dot(c, s), np.dot(b, s))
    return angle_of_rotation*a
```

```
In [208]: checking_vectors = []

          for i, rotation in enumerate(rotations):
              print_array(rotation_matrix_to_vector(rotation), name = "axis-angle vector
          {}".format(i))
              print_array(rotation, name = "rotation matrix {}".format(i))
              print()
```

```
axis-angle vector 0: [0. 0. 0.]
rotation matrix 0:
        [1. 0. 0.]
        [0. 1. 0.]
        [0. 0. 1.]

axis-angle vector 1: [0.524 0.    0.   ]
rotation matrix 1:
        [1. 0. 0.]
        [ 0.     0.866 -0.5  ]
        [0.    0.5   0.866]

axis-angle vector 2: [-1.047 -0.    -0.   ]
rotation matrix 2:
        [1. 0. 0.]
        [0.    0.5   0.866]
        [ 0.    -0.866  0.5  ]

axis-angle vector 3: [0.    1.571 0.   ]
rotation matrix 3:
        [0. 0. 1.]
        [0. 1. 0.]
        [-1.  0.  0.]

axis-angle vector 4: [ 0.    -1.111 -1.111]
rotation matrix 4:
        [-0.     0.707 -0.707]
        [-0.707  0.5    0.5  ]
        [0.707 0.5   0.5  ]
```

Problem 3.3

```
In [209]: def rotation_x(theta):
              c, s = np.cos(theta), np.sin(theta)
              plane_rotation = np.array([[c, -s], [s, c]])
              rx = np.eye(3)
              rx[np.ix_((1, 2), (1, 2))] = plane_rotation
              return rx
```

In [210]:
```
def rotation_vector_to_matrix(r):
    theta = np.linalg.norm(r)
    if theta == 0:
        return np.identity(3)
    xform = rotation_basis(r)
    rx = rotation_x(theta)
    ref_system = xform @ rx @ xform.T
    return ref_system
```

In [200]:
```
axis_angle_pickle = 'angle_axis.pkl'

retrieve(axis_angle_pickle)
with open(axis_angle_pickle, 'rb') as file:
    vectors = pickle.load(file)
```

```
Using previously downloaded file angle_axis.pkl
```

In [213]:
```
for i, vector in enumerate(vectors):
    print_array(vector, name = "axis-angle vector {}".format(i))
    print_array(rotation_vector_to_matrix(vector), name = "rotation matrix {}"
.format(i))
    print()
```

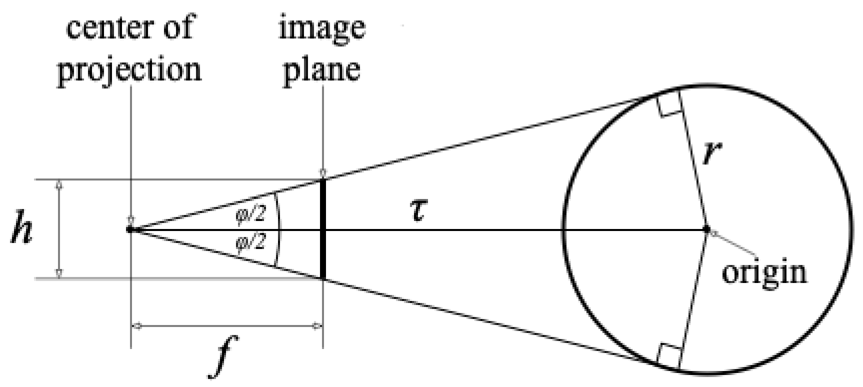
```
axis-angle vector 0: [0. 0. 0.]
rotation matrix 0:
        [1. 0. 0.]
        [0. 1. 0.]
        [0. 0. 1.]

axis-angle vector 1: [0.    0.    1.047]
rotation matrix 1:
        [ 0.5   -0.866  0.   ]
        [0.866 0.5   0.   ]
        [0. 0. 1.]

axis-angle vector 2: [-0.785 -0.    -0.   ]
rotation matrix 2:
        [1. 0. 0.]
        [0.    0.707 0.707]
        [ 0.    -0.707  0.707]

axis-angle vector 3: [1.111 1.111 0.   ]
rotation matrix 3:
        [0.5   0.5   0.707]
        [ 0.5    0.5   -0.707]
        [-0.707  0.707  0.   ]

axis-angle vector 4: [-0.37 -0.   -0.37]
rotation matrix 4:
        [0.933 0.354 0.067]
        [-0.354  0.866  0.354]
        [ 0.067 -0.354  0.933]
```

Part 4: Perspective Projection

Problem 4.1

In [20]:
```python
def transform(p, r=None, t=None):
    if r is None:
        r = np.eye(p.shape[0])
    if t is None:
        t = np.zeros(p.shape[0])
    shifts = np.outer(t, np.ones(p.shape[1]))
    return r @ (p - shifts)
```

In [28]:
```python
radius_mm, distance_mm = 60., 200.
t_mm = distance_mm * np.ones(3, dtype=float) / np.sqrt(3.)
```



In [22]:
```python
def print_camera(model):
    with np.printoptions(precision=3, suppress=True):
        print('Extrinsic parameters:')
        print_array(model.t, 'Origin', 'mm', indent=1)
        print_array(model.R, 'Rotation matrix', indent=1)

        print('\nIntrinsic parameters:')
        print('\tFocal distance: {:.3f} mm'.format(model.f))
        print_array(model.s, 'Scaling', 'pixels per mm', indent=1)
        print_array(model.pi, 'Principal point', 'pixels', indent=1)
```

In [217]:
```
from types import SimpleNamespace

def new_camera(radius, t, pixels, pixel_microns):

    pixels_per_mm = pixels * (1 / pixel_microns) * (10**3)
    principal_point = pixels / 2

    t_mag = np.linalg.norm(t)
    k = -t / t_mag
    i = np.cross(k, np.array([0, 0, 1]))
    j = np.cross(k, i)
    rotation = np.array([i, j, k])


    half_angle = np.arcsin(radius_mm/t_mag)
    half_height = pixels[1]/(2 * pixels_per_mm[1])
    f_mm = half_height/np.tan(half_angle)

    model = SimpleNamespace(R = rotation, t = t, \
                            pixels = pixels, pi = principal_point, \
                            f = f_mm, s = pixels_per_mm)

    return model
```

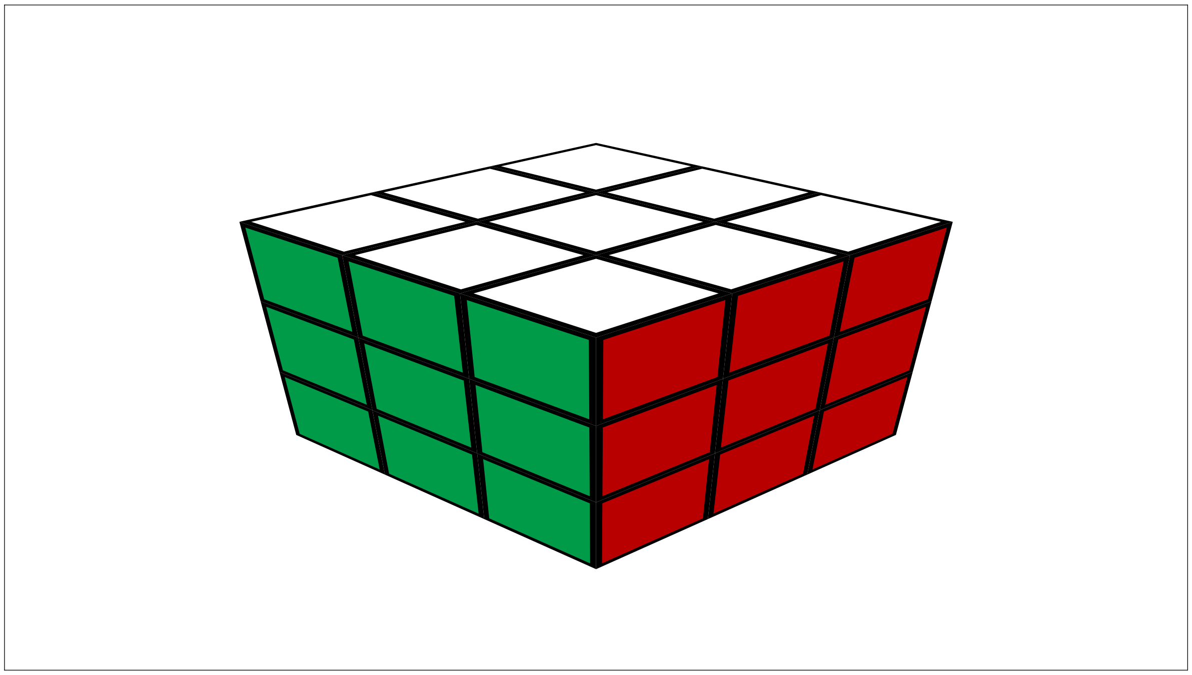
In [218]:
```
print_camera(model = new_camera(radius_mm, t_mm, screen_dims, pixel_size))
```

```
Extrinsic parameters:
        Origin: [115.47 115.47 115.47] mm
        Rotation matrix:
                [-0.577  0.577  0.   ]
                [ 0.333  0.333 -0.667]
                [-0.577 -0.577 -0.577]

Intrinsic parameters:
        Focal distance: 0.008 mm
        Scaling: [384000. 216000.] pixels per mm
        Principal point: [960. 540.] pixels
```

Problem 4.2

In [219]:
```
retrieve('transform.py')
retrieve('rubik.py')
from rubik import new_cube, draw_cube


screen_dims = np.array([1920, 1080])
pixel_size = 5

camera = new_camera(radius_mm, t_mm, screen_dims, pixel_size)
rubik_cube = new_cube()

draw_cube(rubik_cube, camera)
```

```
Using previously downloaded file transform.py
Using previously downloaded file rubik.py
```



In [221]:
```
rear_t_mm = distance_mm * np.array((-1, -1, 2), \
                                   dtype=float) / np.sqrt(6.)
```

```
In [222]: camera_2 = new_camera(radius_mm, rear_t_mm, screen_dims, 5)
          draw_cube(rubik_cube, camera_2)
```