Problem 1. (25 pts) We can treat the human fovea as a square sensor array of size 1.5 mm × 1.5 mm, containing about 337,000 cones (sensor elements). Assume that the space between cones is equal to width of a cone, and that the focal length of the eye is 17 mm.

a. What is the field of view (in degrees) of the human fovea?
Given the width of the image plane of the human fovea is $w = 1.5$ mm and the focal length $f = 17$ mm, the field of view θ is

$$\theta = 2 * \tan^{-1}\left(\frac{w}{2f}\right) = 2 * \tan^{-1}\left(\frac{1.5}{2 \cdot 17}\right) \approx \boxed{5.05 \text{ degrees}}.$$

I am assuming here that the lens in the eye compresses the full human field of view down to this 5 degree window that the fovea can view.

b. Estimate the distance from Brown Hall to the top of South Table Mountain (you can find this using a map, or a webtool such as Microsoft Bing Maps, or Google Earth). What is the minimum size object that you can see with the naked eye on top of the mountain? Can you see a person on top of the mountain? Assume for simplicity that size of the image of the object must cover at least two receptors (cones).
Using Google Maps' "Measure Distance" feature, the distance between Brown Hall and South Table Mountain is approximately 3800 ft or about 1158 m. Given (x, y) coordinates of the image plane and (X, Y, Z) coordinates in the world plane, we can find what height Y above South Table Mountain can be seen by the human eye. We consider about $\sqrt{337,000} \approx 581$ cones per side in the image plane with about a width of $1.5mm/581 \approx 0.0026mm$ per side. Assuming the object should be twice the width of one cone for it to be visible, we need $y \approx 2 \times 0.0026mm$ for an object to be detected. Hence, the minimum size object on top of the mountain that can be seen is

$$Y = \frac{yZ}{f} = \frac{2 \times 0.0026mm \times 1158m}{17mm} \approx \boxed{0.4 \text{ m}}.$$

Given an average human height of 1.65 m, a person can be seen on top of the mountain.

Problem 2. (25 pts) A pool-playing robot uses an overhead camera to determine the positions of the balls on the pool table. Assume that:

- We are using a standard billiard table of size 44" × 88"
- We are using standard 57mm Billiards balls
- We need at least 100 square pixels per ball to reliably determine the identity of each ball
- The center of the ball can be located to a precision of ± one pixels in the image

- We need to locate the ball on the table to an accuracy of ± one cm
- We are going to mount the camera on the ceiling, looking straight down. The distance from the camera to the table is 2m.

Determine a configuration of the camera resolution and lens FOV that will meet these requirements. Assume that you can choose from the following parts:

- Lenses with field of view 30, 60, 90 degrees
- Cameras with resolutions of 256×256, 512×512, or 1024×1024 pixels
- Choose the lowest resolution that will meet the requirements.

To determine which camera parameters meet the imaging requirements, the focal length f was determined in terms of pixels based on the given resolution width w (256, 512, or 1024),

$$f = \frac{w}{2\tan\left(\frac{\theta}{2}\right)},$$

where θ represents the field of view that can be either 30, 60, or 90 degrees. From the focal length f, the dimensions of the table, ball, and location accuracy were mapped to the equivalent pixel amount on the image, as follows,

$$\#\text{ of pixels} = \frac{f \times (\text{Dimension on Billiard Table})}{Z},$$

where $Z = 2m$. The output pixels were then compared against the image requirements (resolution size, 10×10 square pixels per ball, and ± 1 pixel precision) to evaluate which camera settings meet the requirements. The code to calculate all 9 cases is shown below along with the output in Listing 1. The lowest resolution case that meets all imaging requirements is $\boxed{512 \times 512 \text{ pixels}}$ with a lens with field of view of 60 degrees.

```python
# Soraya Terrab - CSCI 507 - Computer Vision - Fall 2020
# Homework 1
# Question 2

import numpy as np
import sys

sys.stdout = open("Homework1_Q2_output.txt", "w")
# Billiard Table/Ball Parameters, dimensions in meters
table_width = 88 * 2.54e-2 # inches converted to meters
table_height = 44 * 2.54e-2
table_distance_from_camera = 2

ball_size = 57 * 1e-3 # mm to m
```

```
ball_location_accuracy = 1e-2

# Image Requirements
num_of_pixels_per_dim = 10
pixel_precision = 1

# Camera Lens Field of View (FOV)
FOV = np.array([30, 60, 90])
Resolution  = np.array ([256, 512, 1024])

for k in range(len(Resolution)):
    print( "\n #### Camera Resolution: " + str(Resolution[k]) + "x" +  ...
        str(Resolution[k]) + " ####" )
    for i in range(len(FOV)):
        print("\n Results for Lens FOV of " +  str(FOV[i])+ ":")
        # Calculating Focal length for given FOV and camera resolution, ...
            converting degrees to radians
        focal_length = Resolution[k]/(2*np.tan((np.pi/180 *FOV[i])/2))

        # Pixels Needed to Image Pool Table
        pixels_for_table_width = focal_length * table_width / ...
            table_distance_from_camera
        pixels_for_table_height = focal_length * table_height / ...
            table_distance_from_camera
        if pixels_for_table_width <= Resolution[k] and ...
            pixels_for_table_height <= Resolution[k]:
             Output = "Meets Table Imaging Requirements"
        else:
             Output = "Does Not Meet Table Imaging Requirements"
        print("¬Pixels Needed to Image the Billiard Table : " + ...
            str(pixels_for_table_height.astype(int)) + "x" + ...
            str(pixels_for_table_width.astype(int)))
        print("¬¬¬¬"+ str(Output))

        # Pixels Needed to Image Billiard Ball
        pixels_for_ball = focal_length * ball_size / ...
            table_distance_from_camera
        if pixels_for_ball >= num_of_pixels_per_dim:
             Output = "Meets Ball Imaging Requirements"
        else:
             Output = "Does Not Meet Ball Imaging Requirements"
        print("¬Pixels Needed to Image the Billiard Ball : " + ...
            str(round(pixels_for_ball,2)) + "x" + ...
            str(round(pixels_for_ball,2)))
        print("¬¬¬¬"+ str(Output))

        # Pixels Needed to Achieve Location Accuracy
        pixels_for_center_location = focal_length * ...
            ball_location_accuracy / table_distance_from_camera
        if pixels_for_center_location >= pixel_precision:
```

```
            Output = "Meets Center Precision for Ball Location Accuracy"
        else:
            Output = "Does Not Meet Center Precision for Ball Location ...
                Accuracy"
        print("¬Pixels Needed for Location Accuracy: " + ...
            str(round(pixels_for_center_location,2)))
        print("¬¬¬¬" + str(Output))

sys.stdout.close()
```

Listing 1: Camera Parameters and Results for Image Requirements

```

 #### Camera Resolution: 256x256 ####

 Results for Lens FOV of 30:
¬Pixels Needed to Image the Billiard Table : 266x533
¬¬¬¬Does Not Meet Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 13.61x13.61
¬¬¬¬Meets Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 2.39
¬¬¬¬Meets Center Precision for Ball Location Accuracy

 Results for Lens FOV of 60:
¬Pixels Needed to Image the Billiard Table : 123x247
¬¬¬¬Meets Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 6.32x6.32
¬¬¬¬Does Not Meet Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 1.11
¬¬¬¬Meets Center Precision for Ball Location Accuracy

 Results for Lens FOV of 90:
¬Pixels Needed to Image the Billiard Table : 71x143
¬¬¬¬Meets Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 3.65x3.65
¬¬¬¬Does Not Meet Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 0.64
¬¬¬¬Does Not Meet Center Precision for Ball Location Accuracy

 #### Camera Resolution: 512x512 ####

 Results for Lens FOV of 30:
¬Pixels Needed to Image the Billiard Table : 533x1067
¬¬¬¬Does Not Meet Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 27.23x27.23
¬¬¬¬Meets Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 4.78
¬¬¬¬Meets Center Precision for Ball Location Accuracy

```

```
 Results for Lens FOV of 60:
¬Pixels Needed to Image the Billiard Table : 247x495
¬¬¬¬Meets Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 12.64x12.64
¬¬¬¬Meets Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 2.22
¬¬¬¬Meets Center Precision for Ball Location Accuracy

 Results for Lens FOV of 90:
¬Pixels Needed to Image the Billiard Table : 143x286
¬¬¬¬Meets Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 7.3x7.3
¬¬¬¬Does Not Meet Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 1.28
¬¬¬¬Meets Center Precision for Ball Location Accuracy

 #### Camera Resolution: 1024x1024 ####

 Results for Lens FOV of 30:
¬Pixels Needed to Image the Billiard Table : 1067x2135
¬¬¬¬Does Not Meet Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 54.46x54.46
¬¬¬¬Meets Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 9.55
¬¬¬¬Meets Center Precision for Ball Location Accuracy

 Results for Lens FOV of 60:
¬Pixels Needed to Image the Billiard Table : 495x991
¬¬¬¬Meets Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 25.27x25.27
¬¬¬¬Meets Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 4.43
¬¬¬¬Meets Center Precision for Ball Location Accuracy

 Results for Lens FOV of 90:
¬Pixels Needed to Image the Billiard Table : 286x572
¬¬¬¬Meets Table Imaging Requirements
¬Pixels Needed to Image the Billiard Ball : 14.59x14.59
¬¬¬¬Meets Ball Imaging Requirements
¬Pixels Needed for Location Accuracy: 2.56
¬¬¬¬Meets Center Precision for Ball Location Accuracy
```

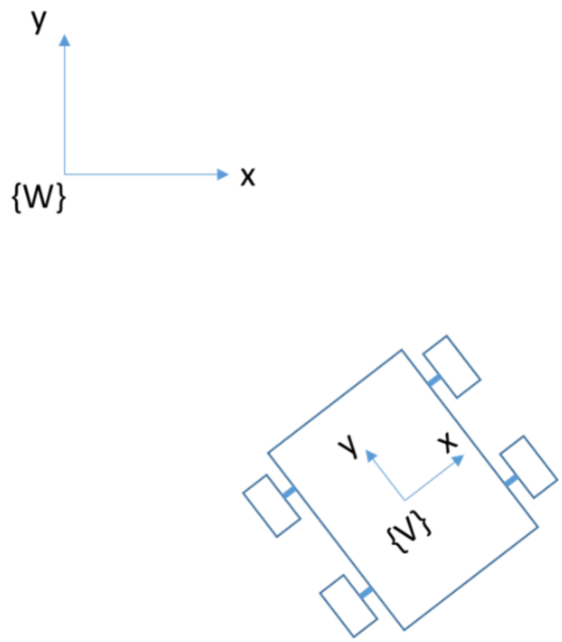

Figure 1: Top View of the vehicle $\{V\}$ in the world frame.

Problem 3. (50 pts) A vehicle $\{V\}$ is positioned at $(6,\ -8,\ 1)$ with respect to the world $\{W\}$. It is rotated by 30 degrees about the world Z axis, which points up. Figure 1 shows a top down view of the scene.
A camera $\{C\}$ is mounted on a rotational mount $\{M\}$ on the vehicle, as shown in Fig. 2. The mount $\{M\}$ is positioned directly above the vehicle origin at a distance = 3. It is tilted down by 30 degrees. The camera is rigidly attached to the mount. It is positioned directly above the mount origin at a distance = 1.5.
A pyramid has vertices in world coordinates: $(-1,\ -1,\ 0)$, $(1,\ -1,\ 0)$, $(1,\ 1,\ 0)$, $(-1,\ 1,\ 0)$, $(0,\ 0,\ 3)$. Using Python, generate an image of a wireframe model of the pyramid as if were seen by the camera, similar to the figure below, and a 3D plot showing the poses of the camera, mount, vehicle, and pyramid. Assume a pinhole camera model, with focal length = 600, where the image size is 640 pixels wide by 480 pixels high. **Hints:** You will have to combine transformations; i.e. calculate the transformation from the camera to the world as ${}^{W}_{C}H = {}^{W}_{V}H\ {}^{V}_{M}H\ {}^{M}_{C}H$. The first two vertices of the pyramid, the ones with the world coordinates $(X,\ Y,\ Z) = (-1,\ -1,\ 0)$ and $(1,\ -1,\ 0)$, project to pixel locations $(x, y) = (170,\ 251)$ and $(267,\ 284)$, rounded to the nearest pixel.

The goal is to achieve a transformation matrix from world to camera, ${}^{C}_{W}H$. To do so, we can find the transformation from camera to world, ${}^{W}_{C}H$, which can be found as ${}^{W}_{C}H = {}^{W}_{V}H\ {}^{V}_{M}H\ {}^{M}_{C}H$, and then invert it. The transformation from ${}^{W}_{V}H$ was found by having a 30 degree or $\pi/6$ radians rotation about the z axis as well as by defining the translation vector as $(6, -8, 1)$. Lines 14-21 in the code listing below shows how ${}^{W}_{V}H$ is constructed. Next,

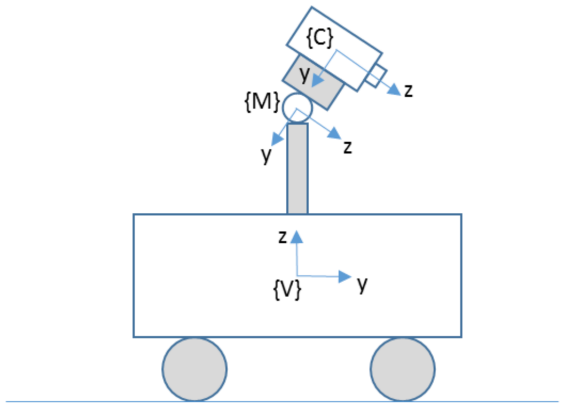

Figure 2: Side View of the camera mounted on the vehicle

the transformation from mount to vehicle ${}^{V}_{M}H$ is considered given a -120 degree (or $-2\pi/3$ radians) rotation about the vehicle z axis. Additionally, the translation vector of the mount in the vehicle frame is $(0, 0, 3)$. Lines 25-32 in the code below are used to construct ${}^{V}_{M}H$. The transformation for camera to mount was set to have an identity rotation matrix (given no rotation) and translation vector $(0, -1.5, 0)$; lines 35-39 below show how ${}^{C}_{M}H$ is constructed. Finally, the product of these transformation matrices is compute to find ${}^{W}_{C}H$ and it is then inverted to determine ${}^{C}_{W}H$. The subsequent steps are similar to the ones used in class as well as Lab 2 to project the 5 pyramid points in the world frame onto an image taken by the camera. Here, $f = 600$, $sx = sy = 1, cx = 320$, $cy = 240$.

As shown in Listing 2 below, the image coordinates of the 5 projected points are:

$$(170,\ 251),\ (267,\ 284),\ (329,\ 230),\ (240,\ 205),\ (241,\ 69).$$

The wireframe image of the projected pyramid and the 3D scene are shown in Figs. 3-4.

```
1 import numpy as np
2 import cv2
3 import sys
4 import matplotlib.pyplot as plt
5 from mpl_toolkits.mplot3d import Axes3D
6
7
8 sys.stdout = open("Homework1_Q3_output.txt", "w")
9
10 # Code Referred from 2_2 (3Dto3DTransforms), 2_3 (3Dto2DTransforms) slides
11
12 ### VEHICLE {V} TO WORLD {W}
13 #Rotation of vehicle relative to world coordinates
14 az = np.pi/6 # 30 degrees in radians
15 sz = np.sin(az)
16 cz = np.cos(az)
```

```
Rz = np.array(((cz, −sz, 0), (sz, cz, 0), (0, 0, 1)))
# Translation: origin of V in W.
tVorg_W = np.array([[6,−8,1]]).T
# H_V_W means transform V to W.
H_V_W = np.block([[Rz, tVorg_W], [0,0,0,1]])

### MOUNT {M} TO VEHICLE {V}
#Rotation of mount relative to vehicle coordinates
ax = − 2*np.pi/3 # −120 degrees in radians
sx= np.sin(ax)
cx = np.cos(ax)
Rx = np.array(((1, 0, 0), (0, cx, −sx), (0, sx, cx)))
# Translation: origin of M in V.
tMorg_V = np.array([[0,0,3]]).T
# H_M_V means transform M to V.
H_M_V = np.block([[Rx, tMorg_V], [0,0,0,1]])

### CAMERA {C} TO MOUNT {M}
I = np.identity(3) # no rotation for this case, just identity matrix
# Translation: origin of C in M.
tCorg_M = np.array([[0,−1.5,0]]).T
# H_C_M means transform C to M.
H_C_M = np.block([[I, tCorg_M], [0,0,0,1]])

### WORLD {W} to CAMERA {C}
H_C_W = H_V_W @ H_M_V @ H_C_M
H_W_C = np.linalg.inv(H_C_W)

### CAMERA
# Intrisic Calibration Matrix
f = 600.0 # focal length in pixels
sx = 1
sy = 1
cx = 320
cy = 240
K = np.array(((f/sx, 0, cx), (0, f/sy, cy), (0, 0, 1)))
# Extrinsic Matrix
Mext = H_W_C[0:3, :]

### PYRAMID
# In World Coordinates
P_w = np.array([[−1, 1, 1, −1, 0],
                [−1, −1, 1, 1, 0],
                [0, 0, 0, 0, 3],
                np.ones(5)])
# In Camera Coordinates
p_C = K @ Mext @ P_w
p_C = p_C / p_C[2,:] # Keeping in Homogeneous Coordinates
print(np.rint(p_C[0:2, :]))

```

```
### Creating Wireframe image of Pyramid
Image = 255*np.ones((480, 640), dtype=np.uint8)

# Adding Wireframe to adjacent points
for i in range(4):
    cv2.line(Image, (p_C[0,i].astype(int), p_C[1,i].astype(int)), ...
        (p_C[0,i+1].astype(int), p_C[1,i+1].astype(int)), 0, thickness=2)

#Adding Wireframe to other 3 base points to vertex of pyramid
cv2.line(Image, (p_C[0,0].astype(int), p_C[1,0].astype(int)), ...
    (p_C[0,4].astype(int), p_C[1,4].astype(int)), 0, thickness=2)
cv2.line(Image, (p_C[0,1].astype(int), p_C[1,1].astype(int)), ...
    (p_C[0,4].astype(int), p_C[1,4].astype(int)), 0, thickness=2)
cv2.line(Image, (p_C[0,2].astype(int), p_C[1,2].astype(int)), ...
    (p_C[0,4].astype(int), p_C[1,4].astype(int)), 0, thickness=2)
# Connecting First and Fourth base corners
cv2.line(Image, (p_C[0,0].astype(int), p_C[1,0].astype(int)), ...
    (p_C[0,3].astype(int), p_C[1,3].astype(int)), 0, thickness=2)

cv2.imshow("Pyramid Wireframe", Image)
cv2.imwrite("Homework1_PyramidWireframe.jpg", Image)
cv2.waitKey(0)


### Plotting 3D scene, Code used from 2-4, TransformsAdditional slides

# Draw three 3D line segments, representing xyz unit axes, onto the ...
    axis figure ax.
# H is the 4x4 transformation matrix representing the pose of the ...
    coordinate frame.
def draw_coordinate_axes(ax, H, label):
    p = H[0:3, 3]      # Origin of the coordinate frame
    ux = H @ np.array([1,0,0,1])# Tip of the x axis
    uy = H @ np.array([0,1,0,1])# Tip of the y axis
    uz = H @ np.array([0,0,1,1])# Tip of the z axis
    ax.plot(xs=[p[0], ux[0]], ys=[p[1], ux[1]], zs=[p[2], ux[2]], ...
        c='r')# x axis
    ax.plot(xs=[p[0], uy[0]], ys=[p[1], uy[1]], zs=[p[2], uy[2]], ...
        c='g')# y axis
    ax.plot(xs=[p[0], uz[0]], ys=[p[1], uz[1]], zs=[p[2], uz[2]], ...
        c='b')# z axis
    ax.text(p[0], p[1], p[2], label)   # Also draw the label of the ...
        coordinate frame

# Utility function for 3D plots.
def setAxesEqual(ax):
    # '''Make axes of 3D plot have equal scale so that spheres appear ...
        as spheres,
    # cubes as cubes, etc..  This is one possible solution to Matplotlib's
    # ax.set_aspect('equal') and ax.axis('equal') not working for 3D.
```

```
    # Input      ax: a matplotlib axis, e.g., as output from plt.gca().
    # '''
    x_limits = ax.get_xlim3d()
    y_limits = ax.get_ylim3d()
    z_limits = ax.get_zlim3d()
    x_range = abs(x_limits[1] - x_limits[0])
    x_middle = np.mean(x_limits)
    y_range = abs(y_limits[1] - y_limits[0])
    y_middle = np.mean(y_limits)
    z_range = abs(z_limits[1] - z_limits[0])
    z_middle = np.mean(z_limits)

    # The plot bounding box is a sphere in the sense of the infinity
    # norm, hence I call half the max range the plot radius.
    plot_radius = 0.5 * max([x_range, y_range, z_range])
    ax.set_xlim3d([x_middle - plot_radius, x_middle + plot_radius])
    ax.set_ylim3d([y_middle - plot_radius, y_middle + plot_radius])
    ax.set_zlim3d([z_middle - plot_radius, z_middle + plot_radius])

#creating Pyramid coordinates such that we can form wireframe using 3D plot
Pyramid = np.c_[P_w, P_w[:,2], P_w[:,4], P_w[:,1], P_w[:,4], P_w[:,0], ...
    P_w[:,3]]

#Creating Figure
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(Pyramid[0,:], Pyramid[1,:], zs=Pyramid[2,:])
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D Scene of Vehicle, Mount, and Camera in World with ...
    Pyramid')
ax.view_init(20, 30)
draw_coordinate_axes(ax, np.eye(4), 'W') #world
draw_coordinate_axes(ax, H_C_W, 'C') # camera pose in world
draw_coordinate_axes(ax, H_V_W, 'V') # vehicle pose in world
draw_coordinate_axes(ax, H_V_W @ H_M_V, 'M') # mount pose in world
setAxesEqual(ax)
fig.savefig("Homework1_3DScene.jpg")
plt.show()  # This shows the plot, and pauses until you close the figure


sys.stdout.close()
```

Listing 2: Printed Output of points in image plane

```
[[170. 267. 329. 240. 241.]
 [251. 284. 230. 205.  69.]]
```

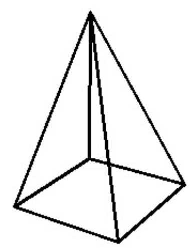

Figure 3: Image taken by Camera of Pyramid

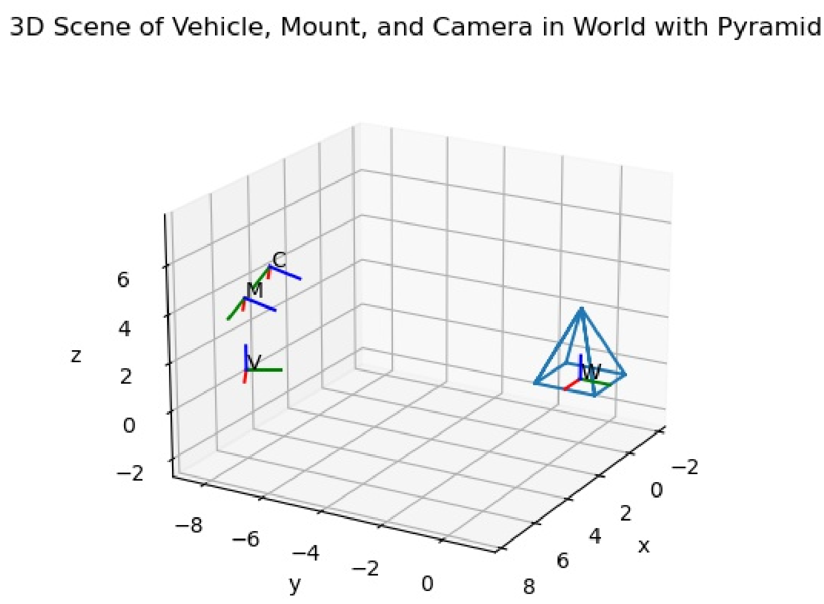

Figure 4: 3D Plot