# Part 3: Single-View Geometry

## Usage

This code snippet provides an overall code structure and some interactive plot interfaces for the *Single-View Geometry* section of Assignment 3. In main function, we outline the required functionalities step by step. Some of the functions which involves interactive plots are already provided, but the rest are left for you to implement.

## Package installation

- In this code, we use `tkinter` package. Installation instruction can be found here (https://anaconda.org/anaconda/tk).

# Common imports

In [86]:
```
% matplotlib tk
import matplotlib.pyplot as plt
import numpy as np

from PIL import Image
```

# Provided functions

In [88]:
```
def get_input_lines(im, min_lines=3):
    """
    Allows user to input line segments; computes centers and directions.
    Inputs:
        im: np.ndarray of shape (height, width, 3)
        min_lines: minimum number of lines required
    Returns:
        n: number of lines from input
        lines: np.ndarray of shape (3, n)
            where each column denotes the parameters of the line equation
        centers: np.ndarray of shape (3, n)
            where each column denotes the homogeneous coordinates of the centers
    """
    n = 0
    lines = np.zeros((3, 0))
    centers = np.zeros((3, 0))

    plt.figure()
    plt.imshow(im)
    plt.show()
    print('Set at least %d lines to compute vanishing point' % min_lines)
    while True:
        print('Click the two endpoints, use the right key to undo, and use the mi
        clicked = plt.ginput(2, timeout=0, show_clicks=True)
        if not clicked or len(clicked) < 2:
            if n < min_lines:
                print('Need at least %d lines, you have %d now' % (min_lines, n)
                continue
            else:
                # Stop getting lines if number of lines is enough
                break

        # Unpack user inputs and save as homogeneous coordinates
        pt1 = np.array([clicked[0][0], clicked[0][1], 1])
        pt2 = np.array([clicked[1][0], clicked[1][1], 1])
        # Get line equation using cross product
        # Line equation: line[0] * x + line[1] * y + line[2] = 0
        line = np.cross(pt1, pt2)
        lines = np.append(lines, line.reshape((3, 1)), axis=1)
        # Get center coordinate of the line segment
        center = (pt1 + pt2) / 2
        centers = np.append(centers, center.reshape((3, 1)), axis=1)

        # Plot line segment
        plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], color='b')

        n += 1

    return n, lines, centers
```

In [87]:
```
def plot_lines_and_vp(im, lines, vp):
    """
    Plots user-input lines and the calculated vanishing point.
    Inputs:
        im: np.ndarray of shape (height, width, 3)
        lines: np.ndarray of shape (3, n)
            where each column denotes the parameters of the line equation
        vp: np.ndarray of shape (3, )
    """
    bx1 = min(1, vp[0] / vp[2]) - 10
    bx2 = max(im.shape[1], vp[0] / vp[2]) + 10
    by1 = min(1, vp[1] / vp[2]) - 10
    by2 = max(im.shape[0], vp[1] / vp[2]) + 10

    plt.figure()
    plt.imshow(im)
    for i in range(lines.shape[1]):
        if lines[0, i] < lines[1, i]:
            pt1 = np.cross(np.array([1, 0, -bx1]), lines[:, i])
            pt2 = np.cross(np.array([1, 0, -bx2]), lines[:, i])
        else:
            pt1 = np.cross(np.array([0, 1, -by1]), lines[:, i])
            pt2 = np.cross(np.array([0, 1, -by2]), lines[:, i])
        pt1 = pt1 / pt1[2]
        pt2 = pt2 / pt2[2]
        plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], 'g')

    plt.plot(vp[0] / vp[2], vp[1] / vp[2], 'ro')
    plt.show()
```

In [89]:
```
def get_top_and_bottom_coordinates(im, obj):
    """
    For a specific object, prompts user to record the top coordinate and the bott
    Inputs:
        im: np.ndarray of shape (height, width, 3)
        obj: string, object name
    Returns:
        coord: np.ndarray of shape (3, 2)
            where coord[:, 0] is the homogeneous coordinate of the top of the obj
            coordinate of the bottom
    """
    plt.figure()
    plt.imshow(im)

    print('Click on the top coordinate of %s' % obj)
    clicked = plt.ginput(1, timeout=0, show_clicks=True)
    x1, y1 = clicked[0]
    # Uncomment this line to enable a vertical line to help align the two coordir
    # plt.plot([x1, x1], [0, im.shape[0]], 'b')
    print('Click on the bottom coordinate of %s' % obj)
    clicked = plt.ginput(1, timeout=0, show_clicks=True)
    x2, y2 = clicked[0]

    plt.plot([x1, x2], [y1, y2], 'b')

    return np.array([[x1, x2], [y1, y2], [1, 1]])
```

# Your implementation

In [90]:
```
def get_vanishing_point(lines):
    """
    Solves for the vanishing point using the user-input lines.
    """
    s = lines.dot(lines.T)
    w, v = np.linalg.eig(s)
    min_index = np.argmin(w)
    point = v[:, min_index]
    point = point / point[-1]
    return point
```

In [91]:
```
def get_horizon_line(points):
    """
    Calculates the ground horizon line.
    """
    p1 = points[:, 0]
    p2 = points[:, 1]
    line = np.cross(p1, p2)
    norm = np.sqrt(line[0] ** 2 + line[1] ** 2)
    return line / norm
```

In [92]:
```
def plot_horizon_line(image, line):
    """
    Plots the horizon line.
    """
    r = image.shape[1]
    x = np.arange(r)
    y = (-line[2] - line[0] * x) / line[1]
    plt.figure()
    plt.imshow(image)
    plt.plot(x, y)
    plt.show()
```

In [93]:
```
from sympy import solve, symbols, Matrix, Symbol


def get_camera_parameters(vpts):
    """
    Computes the camera parameters. Hint: The SymPy package is suitable for this.
    """
    f = Symbol('f')
    u = Symbol('u')
    v = Symbol('v')
    v1 = Matrix(vpts[:, 0])
    v2 = Matrix(vpts[:, 1])
    v3 = Matrix(vpts[:, 2])

    inverse_K = Matrix(((f, 0, u), (0, f, v), (0, 0, 1))).inv()

    e12 = v1.T * inverse_K.T * inverse_K * v2
    e13 = v1.T * inverse_K.T * inverse_K * v3
    e23 = v2.T * inverse_K.T * inverse_K * v3
    sol = solve([e12, e13, e23], [f, u, v])
    f = sol[0][0]
    u = sol[0][1]
    v = sol[0][2]
    return abs(f), u, v
```

In [94]:
```
def get_rotation_matrix(vpts, K):
    """
    Computes the rotation matrix using the camera parameters.
    """
    Y_dir = vpts[:, 2][:, np.newaxis]
    X_dir = vpts[:, 1][:, np.newaxis]
    Z_dir = vpts[:, 0][:, np.newaxis]

    inverse_K = np.array(K.inv()).astype(np.float)

    r1 = inverse_K.dot(X_dir)
    r2 = inverse_K.dot(Y_dir)
    r3 = inverse_K.dot(Z_dir)

    r1 = r1 / np.linalg.norm(r1)
    r2 = r2 / np.linalg.norm(r2)
    r3 = r3 / np.linalg.norm(r3)

    R = np.concatenate((r1, r2, r3), axis=1)
    return R
```

In [95]:
```
def estimate_height(vpts, reference_height, reference_coord, obj_coord, horizon_l
    """
    Estimates height for a specific object using the recorded coordinates. You mi
    your report.
    """
    p = vpts[:, 2]
    t0 = reference_coord[:, 0]
    b0 = reference_coord[:, 1]

    r = obj_coord[:, 0]
    b = obj_coord[:, 1]

    v = np.cross(np.cross(b0, b), horizon_line)
    v = v / v[-1]

    t = np.cross(np.cross(v, t0), np.cross(r, b))
    t = t / t[-1]

    height = reference_height * (np.linalg.norm(r - b) * np.linalg.norm(p - t) /
                                 np.linalg.norm(t - b) / np.linalg.norm(p - r))

    return height
```

# Main function

In [96]:

```python
im = np.asarray(Image.open('CSL.jpeg'))

# Part 1
# Get vanishing points for each of the directions
num_vpts = 3
vpts = np.zeros((3, num_vpts))
for i in range(num_vpts):
    print('Getting vanishing point %d' % i)
    # Get at least three lines from user input
    n, lines, centers = get_input_lines(im)
    # <YOUR IMPLEMENTATION> Solve for vanishing point
    vpts[:, i] = get_vanishing_point(lines)
    print("coordinates: ", vpts[:, i])
    # Plot the lines and the vanishing point
    plot_lines_and_vp(im, lines, vpts[:, i])

# <YOUR IMPLEMENTATION> Get the ground horizon line
horizon_line = get_horizon_line(vpts)
print(horizon_line)
# <YOUR IMPLEMENTATION> Plot the ground horizon line
plot_horizon_line(im, horizon_line)
```

```
Getting vanishing point 0
Set at least 3 lines to compute vanishing point
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
coordinates:  [-235.30547912  212.01324009    1.        ]
Getting vanishing point 1
Set at least 3 lines to compute vanishing point
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
coordinates:  [1.31075114e+03 2.19943040e+02 1.00000000e+00]
Getting vanishing point 2
Set at least 3 lines to compute vanishing point
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
Click the two endpoints, use the right key to undo, and use the middle key to s
top input
```

```
coordinates:  [4.93551618e+02 1.21843891e+04 1.00000000e+00]
[-5.12898158e-03  9.99986847e-01 -2.13217329e+02]


C:\Users\rensy\AppData\Local\Temp\ipykernel_12480\454980637.py:24: RuntimeWarni
ng: divide by zero encountered in true_divide
  pt1 = pt1 / pt1[2]
C:\Users\rensy\AppData\Local\Temp\ipykernel_12480\454980637.py:24: RuntimeWarni
ng: invalid value encountered in true_divide
  pt1 = pt1 / pt1[2]
C:\Users\rensy\AppData\Local\Temp\ipykernel_12480\454980637.py:25: RuntimeWarni
ng: divide by zero encountered in true_divide
  pt2 = pt2 / pt2[2]
C:\Users\rensy\AppData\Local\Temp\ipykernel_12480\454980637.py:25: RuntimeWarni
ng: invalid value encountered in true_divide
  pt2 = pt2 / pt2[2]
```

In [97]:

```python
# Part 2
# <YOUR IMPLEMENTATION> Solve for the camera parameters (f, u, v)
f, u, v = get_camera_parameters(vpts)
print("f = {}, u = {}, v = {}".format(f, u, v))

# Part 3
# <YOUR IMPLEMENTATION> Solve for the rotation matrix
K = Matrix(((f, 0, u), (0, f, v), (0, 0, 1)))
R = get_rotation_matrix(vpts, K)
print(R)

# Part 4
# Record image coordinates for each object and store in map
objects = ('person', 'CSL building', 'the spike statue', 'the lamp posts')
coords = dict()
for obj in objects:
    coords[obj] = get_top_and_bottom_coordinates(im, obj)

# <YOUR IMPLEMENTATION> Estimate heights
reference_height = 66.0
# reference_height = 1.6764
for obj in objects[1:]:
    print('Estimating height of %s' % obj)
    height = estimate_height(vpts, reference_height, coords['person'], coords[obj
    print("Height of {} is {}".format(obj, height))

reference_height = 72.0
# reference_height = 1.8288
for obj in objects[1:]:
    print('Estimating height of %s' % obj)
    height = estimate_height(vpts, reference_height, coords['person'], coords[obj
    print("Height of {} is {}".format(obj, height))
```

```
f = 771.233799695055, u = 554.681767883752, v = 265.971208360832
[[ 0.69941661 -0.00511828 -0.71469589]
 [-0.04257925  0.99789983 -0.04881539]
 [ 0.71344476  0.06457351  0.69772978]]
Click on the top coordinate of person

C:\Users\rensy\AppData\Local\Temp\ipykernel_12480\2929713052.py:9: DeprecationW
arning: `np.float` is a deprecated alias for the builtin `float`. To silence th
is warning, use `float` by itself. Doing this will not modify any behavior and
is safe. If you specifically wanted the numpy scalar type, use `np.float64` her
e.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devd
ocs/release/1.20.0-notes.html#deprecations (https://numpy.org/devdocs/release/
1.20.0-notes.html#deprecations)
  inverse_K = np.array(K.inv()).astype(np.float)

Click on the bottom coordinate of person
Click on the top coordinate of CSL building
Click on the bottom coordinate of CSL building
Click on the top coordinate of the spike statue
Click on the bottom coordinate of the spike statue
Click on the top coordinate of the lamp posts
Click on the bottom coordinate of the lamp posts
```

```
Estimating height of CSL building
Height of CSL building is 1041.235449646049
Estimating height of the spike statue
Height of the spike statue is 358.99447359410595
Estimating height of the lamp posts
Height of the lamp posts is 164.47121356304896
Estimating height of CSL building
Height of CSL building is 1135.8932177956897
Estimating height of the spike statue
Height of the spike statue is 391.63033482993376
Estimating height of the lamp posts
Height of the lamp posts is 179.4231420687807
```

In [98]:

```
objects = ('person', 'person2', 'person3', 'person4')
coords = dict()
for obj in objects:
    coords[obj] = get_top_and_bottom_coordinates(im, obj)

# <YOUR IMPLEMENTATION> Estimate heights
reference_height = 66.0
# reference_height = 1.6764
for obj in objects[1:]:
    print('Estimating height of %s' % obj)
    height = estimate_height(vpts, reference_height, coords['person'], coords[obj
    print("Height of {} is {}".format(obj, height))
```

```
Click on the top coordinate of person
Click on the bottom coordinate of person
Click on the top coordinate of person2
Click on the bottom coordinate of person2
Click on the top coordinate of person3
Click on the bottom coordinate of person3
Click on the top coordinate of person4
Click on the bottom coordinate of person4
Estimating height of person2
Height of person2 is 56.22517429917324
Estimating height of person3
Height of person3 is 41.718546216727624
Estimating height of person4
Height of person4 is 54.53714182078599
```

In [ ]: