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.

Common imports

In [ ]:
```
%matplotlib tk
import matplotlib.pyplot as plt
import numpy as np
from sympy import *
from sympy import solve

from PIL import Image
```

Provided functions

In [ ]:
```
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))
    endpoints = np.zeros((6-0))

    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 middl
        clicked = plt.ginput(2, timeout=0, show_clicks=True)
        if not clicked or len(clicked) < 2:
```

```python
            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)
        endpoint = np.hstack((pt1, pt2))
        endpoints = np.append(endpoints, endpoint.reshape((6, 1)), axis=1)

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

        n += 1

    return n, lines, centers, endpoints
```

In [ ]:
```python
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 [ ]:
```python
def get_top_and_bottom_coordinates(im, obj):
    """
    For a specific object, prompts user to record the top coordinate and the bottom c
    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 object
            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 coordinates
    # 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 [ ]:
```
def get_vanishing_point(n, lines, centers, endpoints):
    """
    Solves for the vanishing point using the user-input lines.
    """
    # <YOUR IMPLEMENTATION>
    bestscore = 0
    sigma = 0.1
    point = np.zeros((3,1))
    for i in range(n):
        for j in range(i+1, n):
            point = np.cross(lines[:,i], lines[:,j])
            if not point[-1]==0:
                score = 0
                x1 = endpoints[:,i][0]
                y1 = endpoints[:,i][1]
                x2 = endpoints[:,i][3]
                y2 = endpoints[:,i][4]
                angle = np.arctan2(abs(y2-y1), abs(x2-x1))
                angle = (angle+np.pi) % 2*np.pi - np.pi
                length = np.linalg.norm([abs(y2-y1), abs(x2-x1)])
                score += length * np.exp(-abs(angle)/(2*sigma**2))
                x1 = endpoints[:,j][0]
                y1 = endpoints[:,j][1]
                x2 = endpoints[:,j][3]
                y2 = endpoints[:,j][4]
                angle = np.arctan2(abs(y2-y1), abs(x2-x1))
                length = np.linalg.norm([abs(y2-y1), abs(x2-x1)])
                score += length * np.exp(-abs(angle)/(2*sigma**2))

                if score > bestscore:
                    score = bestscore
                    bestpoint = point/point[-1]
    return bestpoint
    pass
```

In [ ]:
```
def get_horizon_line(vpts):
    """
    Calculates the ground horizon line.
```

```
    """
    # <YOUR IMPLEMENTATION>
    horizon_line = np.cross(vpts[:, 0], vpts[:, 1])
    scale = np.linalg.norm([horizon_line[0], horizon_line[1]])
    horizon_line = horizon_line/scale
    return horizon_line
    pass
```

In [ ]:
```
def plot_horizon_line(im, horizon_line):
    """
    Plots the horizon line.
    """
    # <YOUR IMPLEMENTATION>
    col = im.shape[1]
    x_array = np.arange(0, col, 1)
    y_array = horizon_line[0]*x_array+horizon_line[2] / (-horizon_line[1])
    plt.figure()
    plt.imshow(im)
    plt.plot(x_array, y_array, 'g')
    pass
```

In [ ]:
```
def get_camera_parameters(vpts):
    """
    Computes the camera parameters. Hint: The SymPy package is suitable for this.
    """
    # <YOUR IMPLEMENTATION>
    vpt1 = vpts[:, 0][:, np.newaxis]
    vpt2 = vpts[:, 1][:, np.newaxis]
    vpt3 = vpts[:, 2][:, np.newaxis]

    f, px, py= symbols('f, px, py')
    KT = Matrix([[1/f, 0, 0], [0, 1/f, 0], [-px/f, -py/f, 1]])
    K = Matrix([[1/f, 0, -px/f], [0, 1/f, -py/f], [0, 0, 1]])

    eq1 = vpt1.T * KT * K * vpt2
    eq2 = vpt1.T * KT * K * vpt3
    eq3 = vpt2.T * KT * K * vpt3
    f, px, py = solve([eq1[0], eq2[0], eq3[0]], (f, px, py))[0]

    return abs(f), px, py
    pass
```

In [ ]:
```
def get_rotation_matrix(f, u, v, vpts):
    """
    Computes the rotation matrix using the camera parameters.
    """
    # <YOUR IMPLEMENTATION>
    vpt1 = vpts[:, 0][:, np.newaxis]
    vpt2 = vpts[:, 1][:, np.newaxis]
    vpt3 = vpts[:, 2][:, np.newaxis]
    K = np.array([[f, 0, u], [0, f, v], [0, 0, 1]]).astype(np.float64)
    K_inv = np.linalg.inv(K)

    r1 = K_inv.dot(vpt2)
    r2 = K_inv.dot(vpt3)
    r3 = K_inv.dot(vpt1)
    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 [ ]: def estimate_height(coords, obj, horizon_line, vpts):
            """
            Estimates height for a specific object using the recorded coordinates. You might
            your report.
            """
            # <YOUR IMPLEMENTATION>
            horizon_line = horizon_line/np.linalg.norm([horizon_line[0], horizon_line[1]])
            lamp = coords['lamp']
            lamp_top = lamp[:,0]
            lamp_bottom = lamp[:,1]
            object = coords[obj]
            object_top = object[:,0]
            object_bottom = object[:,1]
            bottom_line = np.cross(lamp_bottom, object_bottom)
            vanishing_point = np.cross(bottom_line, horizon_line)
            vanishing_point = vanishing_point/vanishing_point[-1]
            object_line = np.cross(object_bottom, object_top)
            lamptop_vanish = np.cross(lamp_top, vanishing_point)
            target_point = np.cross(lamptop_vanish, object_line)
            target_point = target_point/target_point[-1]
            infinite_vpt = vpts[:,2]
            p1_p3 = np.linalg.norm(object_bottom-object_top)
            p2_p4 = np.linalg.norm(infinite_vpt-target_point)
            p3_p4 = np.linalg.norm(object_top-infinite_vpt)
            p1_p2 = np.linalg.norm(object_bottom-target_point)
            ratio = p1_p3*p2_p4 / (p1_p2*p3_p4)

            plt.figure()
            plt.imshow(im)
            col = im.shape[1]
            x_array = np.arange(0, col, 1)
            y_array = horizon_line[0]*x_array+horizon_line[2] / (-horizon_line[1])
            plt.plot(x_array, y_array, 'g')
            plt.plot([vanishing_point[0], lamp_bottom[0]], [vanishing_point[1], lamp_bottom[
            plt.plot([vanishing_point[0], target_point[0]], [vanishing_point[1], target_poin
            plt.plot([vanishing_point[0], object_top[0]], [vanishing_point[1], object_top[1]
            plt.plot([lamp_top[0], lamp_bottom[0]], [lamp_top[1], lamp_bottom[1]], 'b')
            plt.plot([object_bottom[0], object_top[0]], [object_bottom[1], object_top[1]], '
            plt.plot(vanishing_point[0], vanishing_point[1], 'go')
            plt.show()
            return ratio
            pass
```

Main function

```
In [ ]: 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, endpoints = get_input_lines(im)
            # <YOUR IMPLEMENTATION> Solve for vanishing point
            vpts[:, i] = get_vanishing_point(n, lines, centers, endpoints)
            # Plot the lines and the vanishing point
            #plot_lines_and_vp(im, lines, vpts[:, i])
```

```
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 stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
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 stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
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 stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
Click the two endpoints, use the right key to undo, and use the middle key to stop i
nput
```

In [ ]:

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

In [ ]:

```
# Part 2
# <YOUR IMPLEMENTATION> Solve for the camera parameters (f, u, v)
f, u, v = get_camera_parameters(vpts)
```

In [ ]:

```
# Part 3
# <YOUR IMPLEMENTATION> Solve for the rotation matrix
R = get_rotation_matrix(f, u, v, vpts)
```

In [ ]:

```
# Part 4
# Record image coordinates for each object and store in map
objects = ('lamp', 'front gable', 'side gable')
coords = dict()
for obj in objects:
    coords[obj] = get_top_and_bottom_coordinates(im, obj)


# <YOUR IMPLEMENTATION> Estimate heights
for obj in objects[1:]:
    print('Estimating height of %s' % obj)
```

```python
    height = estimate_height(coords, obj, horizon_line, vpts)
    print(f"height of {obj} is {height} times lamp height")
```

```
Click on the top coordinate of lamp
Click on the bottom coordinate of lamp
Click on the top coordinate of front gable
Click on the bottom coordinate of front gable
Click on the top coordinate of side gable
Click on the bottom coordinate of side gable
Estimating height of front gable
height of front gable is 3.992703724086958 times lamp height
Estimating height of side gable
height of side gable is 3.804148717579115 times lamp height
```