# RojasPelliccia_Maximo_a4_p2

November 19, 2024

# 1 Part 3: Single-View Geometry

## 1.1 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.

## 1.2 Package installation

- In this code, we use `tkinter` package. Installation instruction can be found here.

# 2 Common imports

[12]:
```
%matplotlib tk
import matplotlib.pyplot as plt
import numpy as np

from PIL import Image
```

# 3 Provided functions

[13]:
```
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␣
↪middle key to stop input')
        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
```

[14]:
```
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, )
    """
```

```python
    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()
```

[15]:
```python
def get_top_and_bottom_coordinates(im, obj):
    """
    For a specific object, prompts user to record the top coordinate and the␣
 ↪bottom coordinate in the image.
    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 and coord[:, 1] is the homogeneous
            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]])
```

## 4 Your implementation

[16]:
```
def get_vanishing_point(lines):
    """
    Solves for the vanishing point using the user-input lines.
    """
    # <YOUR IMPLEMENTATION>
    A = lines.T
    _, _, V = np.linalg.svd(A)
    vp = V[-1]
    print(f"Vanishing point: {vp}")
    return vp / vp[2]
    #pass
```

[17]:
```
def get_horizon_line(vpts):
    """
    Calculates the ground horizon line.
    """
    # <YOUR IMPLEMENTATION>
    horizon_line = np.cross(vpts[:, 0], vpts[:, 2])
    print(f"a, b, c: {horizon_line}")
    norm = np.sqrt(horizon_line[0]**2 + horizon_line[1]**2)
    print(f"a^2 + b^2 = 1: {horizon_line/norm}")
    return horizon_line / horizon_line[2]
    #pass
```

[18]:
```
def plot_horizon_line(im, horizon_line):
    """
    Plots the horizon line.
    """
    # <YOUR IMPLEMENTATION>
    x_vals = np.array([0, im.shape[1]])  # Image width bounds
    y_vals = -(horizon_line[0] * x_vals + horizon_line[2]) / horizon_line[1]  #␣
  ↪Solve for y
    plt.figure()
    plt.imshow(im)
    plt.plot(x_vals, y_vals, 'r-', label='Horizon Line')
    plt.legend()
    plt.show()
    #pass
```

[19]:
```python
def get_camera_parameters(vpts):
    """
    Computes the camera parameters. Hint: The SymPy package is suitable for␣
 ↪this.
    """
    # <YOUR IMPLEMENTATION>
    vp1, vp2, vp3 = vpts[:, 0], vpts[:, 1], vpts[:, 2]
    u = (vp1[0] + vp2[0]) / 2
    v = (vp1[1] + vp2[1]) / 2
    f = np.sqrt(-np.dot((vp1[:2] - [u, v]), (vp2[:2] - [u, v])))
    return f, u, v
    #pass
```

[1]:
```python
def get_rotation_matrix(vpts, f, u, v):
    """
    Computes the rotation matrix using the camera parameters.
    """
    # <YOUR IMPLEMENTATION>
    K_inv = np.linalg.inv(np.array([[f, 0, u], [0, f, v], [0, 0, 1]]))
    directions = np.dot(K_inv, vpts)
    R = directions / np.linalg.norm(directions, axis=0)
    return R
```

[21]:
```python
def estimate_height(coords, ref_coords, ref_height, horizon_line, ):
    """
    Estimates height for a specific object using the recorded coordinates. You␣
 ↪might need to plot additional images here for
    your report.
    """
    # <YOUR IMPLEMENTATION>

    obj_top, obj_bottom = coords[:, 0], coords[:, 1]
    ref_top, ref_bottom = ref_coords[:, 0], ref_coords[:, 1]

    object_ground_point = np.cross(obj_bottom, horizon_line)
    reference_ground_point = np.cross(ref_bottom, horizon_line)

    object_ground_point /= object_ground_point[2]
    reference_ground_point /= reference_ground_point[2]

    ref_ground_distance = np.linalg.norm(ref_bottom[:2] -␣
 ↪reference_ground_point[:2])
    obj_ground_distance = np.linalg.norm(obj_bottom[:2] - object_ground_point[:
 ↪2])

    ref_image_height = np.linalg.norm(ref_top[:2] - ref_bottom[:2])
    obj_image_height = np.linalg.norm(obj_top[:2] - obj_bottom[:2])
```

```
    adjusted_ground_ratio = obj_ground_distance / ref_ground_distance
    adjusted_height_ratio = obj_image_height / ref_image_height
    object_height = ref_height * adjusted_height_ratio * adjusted_ground_ratio

    perspective_correction = (obj_ground_distance + ref_ground_distance) / (2 *␣
↪ref_ground_distance)
    object_height *= perspective_correction

    return object_height
    #pass
```

# 5 Main function

[22]:
```
im = np.asarray(Image.open('ECEB.jpg'))

# 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)
    # 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)
# <YOUR IMPLEMENTATION> Plot the ground horizon line
plot_horizon_line(im, horizon_line)

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

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

print(f"Rotation matrix: {R}")

# Part 4
# Record image coordinates for each object and store in map
```

```python
objects = ('person', 'building left side', 'building right side', 'the lamp␣
↪posts')
coords = dict()
for obj in objects:
    coords[obj] = get_top_and_bottom_coordinates(im, obj)
    print(coords[obj])

# <YOUR IMPLEMENTATION> Estimate heights
for obj in objects[1:]:
    print('Estimating height of %s' % obj)
    height = estimate_height(coords[obj], coords['person'], 5.6, horizon_line)
    print('Estimated height of %s: %.2f feet' % (obj, height))
```

```
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 input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Vanishing point: [-9.68195309e-01 -2.50195399e-01 -3.24805549e-04]
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 input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
```

```
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Vanishing point: [ 0.15270808 -0.98827061 -0.00120591]
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 input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Click the two endpoints, use the right key to undo, and use the middle key to
stop input
Vanishing point: [8.37500456e-01 5.46436189e-01 6.91326697e-04]
a, b, c: [-2.01238624e+01 -1.76940648e+03  1.42294733e+06]
a^2 + b^2 = 1: [-1.13724920e-02 -9.99935331e-01  8.04142704e+02]
Rotation matrix: [[ 0.70701806 -0.70701806 -0.13746947]
 [-0.01120082  0.01120082 -0.00286263]
 [ 0.70710678  0.70710678  0.99050187]]
Click on the top coordinate of person
Click on the bottom coordinate of person
[[1.32131853e+03 1.32550342e+03]
 [8.04841449e+02 9.84791992e+02]
 [1.00000000e+00 1.00000000e+00]]
Click on the top coordinate of building left side
Click on the bottom coordinate of building left side
[[385.99419821 356.69992391]
 [206.40127449 855.06020544]
 [  1.           1.        ]]
Click on the top coordinate of building right side
```

```
Click on the bottom coordinate of building right side
[[1.75027040e+03 1.78374957e+03]
 [2.21048412e+02 8.02749001e+02]
 [1.00000000e+00 1.00000000e+00]]
Click on the top coordinate of the lamp posts
Click on the bottom coordinate of the lamp posts
[[810.76117558 817.03852007]
 [478.41953586 894.81672056]
 [  1.           1.        ]]
Estimating height of building left side
Estimated height of building left side: 26.94 feet
Estimating height of building right side
Estimated height of building right side: 21.21 feet
Estimating height of the lamp posts
Estimated height of the lamp posts: 10.78 feet
```

[23]:

```python
for obj in objects[1:]:
    print('Estimating height of %s' % obj)
    height = estimate_height(coords[obj], coords['person'], 6.0, horizon_line)
    print('Estimated height of %s: %.2f feet' % (obj, height))
```

```
Estimating height of building left side
Estimated height of building left side: 28.87 feet
Estimating height of building right side
Estimated height of building right side: 22.72 feet
Estimating height of the lamp posts
Estimated height of the lamp posts: 11.55 feet
```