MP01: Image Formation and Processing

Class	Computer Vision
Type	Asssignment

Problem 1: Computer Vision Problems

1.1. Given a lung CT scan, determining whether the patient has cancer or not

- World state (w): Presence of cancer

  → Discrete: Takes 2 values (Yes or No)
- Data (x): Lung CT scan
- Inference: Classification

1.2. Determining the pose of the human body given an image of the body.

- World state (w): Pose of the human body

  Continuous: joints angles and positions, etc.
- Data (x): Image of the human body
- Inference: Regression

1.3. Determining whether two images of faces match (face verification).

- World state (w): Match or mismatch of 2 faces

  Discrete: Takes 2 values (Yes or No)
- Data (x): Two images of faces
- Inference: Classification

1.4. Determining the 3D position of a point given the positions to which it projects in two cameras at different positions in the world (stereo reconstruction).

- World state (w): 3D position of the point

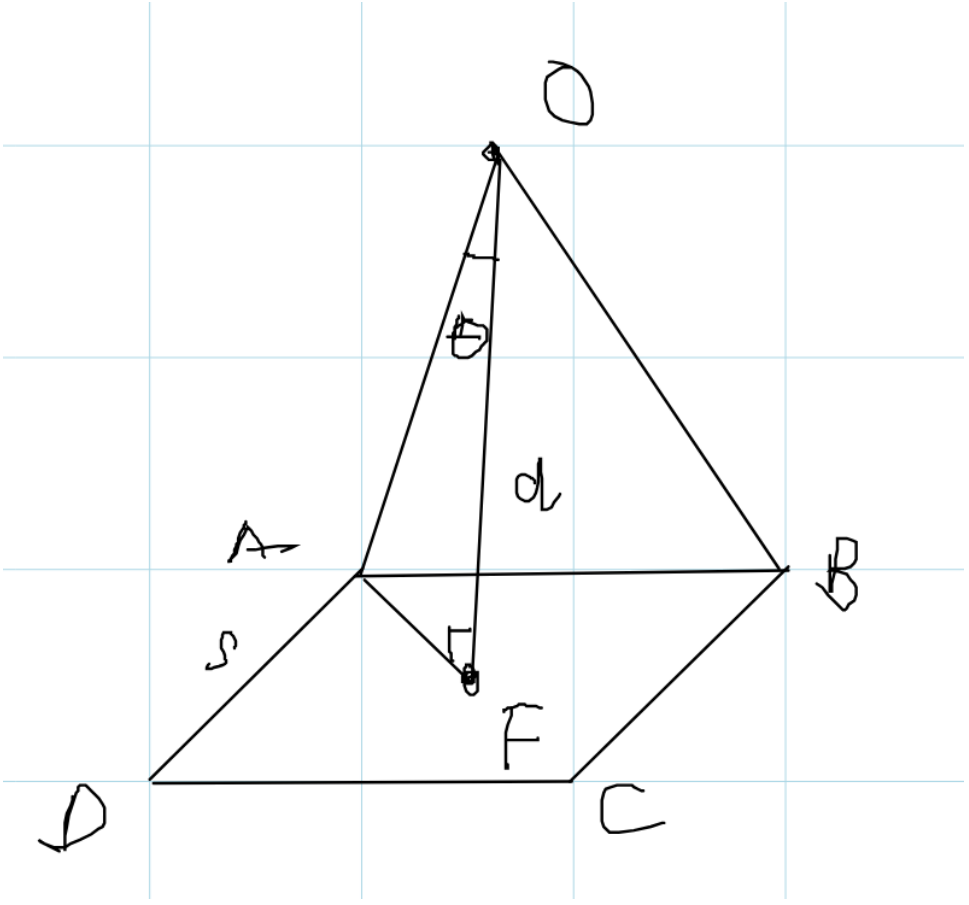
  Continuous: x,y,z coordinates (because x,y,z are continuous)
- Data (x): Positions of the point in two camera images
- Inference: Regression

Problem 2: The Pinhole Camera

We have:

- Sensor size: 1cm x 1cm
- Horizontal FOV: 60 degrees

Diagram (Assuming ideal model)



Let ABCD be the sensor and the focal point is F, O being the optical center

s: width of the sensor (1cm)

d: distance between optical center (O) and sensor (F)

The horizontal FOV is 60 degrees, so the angle θ (between OF and OA) is 30

Assumed that OF is perpendicular to the ABCD plane, we consider the right triangle AOF:

$$tan(\theta) = \frac{AF}{OF} = \frac{s/2}{d}$$

From which

$$d = \frac{s/2}{tan(30)} = \frac{1cm/2}{tan(30)} \approx 0.866cm$$

Problem 3: Image Pixel Manipulation

3.1. Load, display image and show dimensions

Code:

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))#conver to
plt.title('vinuni-campus')
plt.axis('off')
plt.show()

# Dimensions
height, width, channels = image.shape
print(f"Image Dimensions: Width = {width}, Height = {height},
```

Output:


```
Image Dimensions: Width = 1536, Height = 2048, Channels = 3
```

3.2. Access and print pixel values

Code

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

#Access pixel
pixel_value = image[50, 50]

#print value
print(f"Pixel value at (50, 50): {pixel_value}")
```

Output:

```
Pixel value at (50, 50): [146 177 200]
```

3.3. Grayscale image

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

# gray
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# display
plt.imshow(gray_image, cmap='gray')
plt.title('Grayscale Image')

plt.axis('off')
plt.show()
```

Output:

3.4. Image Scaling

Function for scaling, using `cv2.resize()`

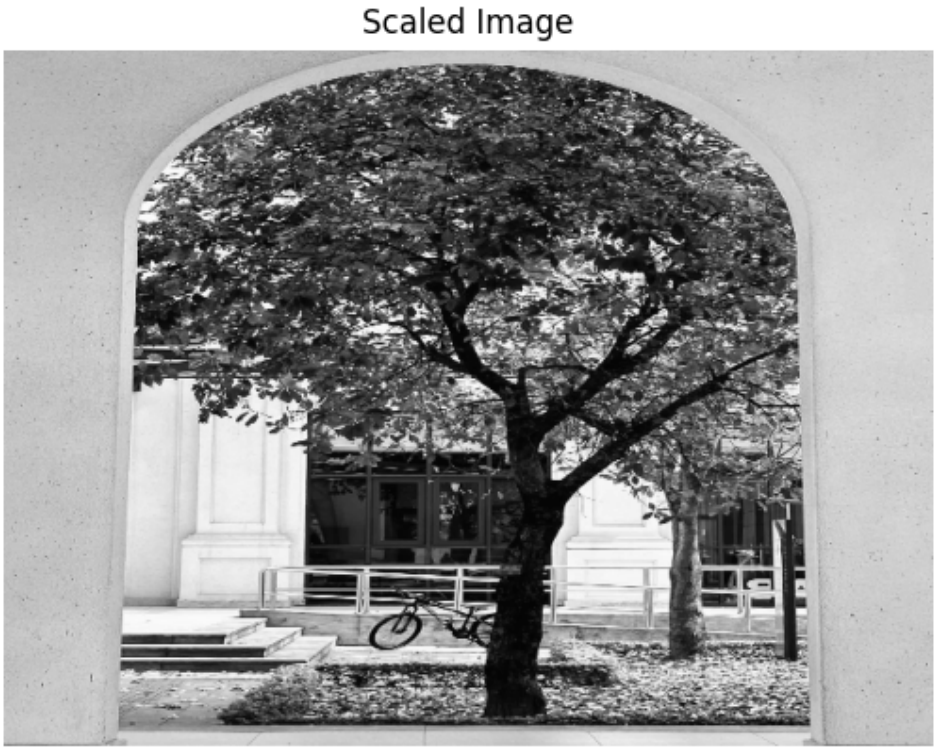
```
def scale_image(img, new_width, new_height):
    scaled_image = cv2.resize(img, (new_width, new_height), i
    return scaled_image
```

Implementation with example:

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

def scale_image(img, new_width, new_height):
    scaled_image = cv2.resize(img, (new_width, new_height), i
```

```
    return scaled_image

# Example
new_width = 640
new_height = 480
scaled_image = scale_image(gray_image, new_width, new_height)

plt.imshow(scaled_image, cmap='gray')
plt.title('Scaled Image')
plt.axis('off')
plt.show()
```

Output



3.5. Find darkest pixel

```
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Find min pixel value and its leocation
min_value = gray_image.min()
min_index = np.unravel_index(gray_image.argmin(), gray_image.

# result
print(f"Darkest pixel value: {min_value}")
print(f"Darkest pixel location (row, column): {min_index}")
```

Output

```
Darkest pixel value: 0
Darkest pixel location (row, column): (np.int64(1383), np.int
```

Problem 4: Gaussian Filtering for Image Smoothing and Noise Reduction

4.1. Gaussian filtering

For this purpose, I am using a prebuilt `cv2.GaussianBlur()` function from OpenCV.

Example usage with provided kernel size and SD:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

# Gaussian Smoothing
kernel_size = (5, 5)
sigma = 1.0
smoothed_image = cv2.GaussianBlur(image, kernel_size, sigma)
```

```
# Display
plt.imshow(cv2.cvtColor(smoothed_image, cv2.COLOR_BGR2RGB))
plt.title('Smoothed Image')
plt.axis('off')
plt.show()
```

Output:

4.2. Adding Gaussion Noise

For this task, I will use Numpy to add random number

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('vinuni_campus.jpg')

def add_gaussian_noise(img, mean=0, std_dev=20):
```

```
    noise = np.random.normal(mean, std_dev, img.shape)
    noisy_image = np.clip(img + noise, 0, 255).astype(np.uint
    return noisy_image

# add noise
noisy_image = add_gaussian_noise(image, std_dev=80)  # std_de

# display
plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
plt.title('Noisy Image')
plt.axis('off')
plt.show()
```

Output:

Findings:

1. Effects of Gaussian Smoothing

- Visual change: Compare to the original image, the Gaussian smoothed image is quite blurry which can be observed around the edges, they are less sharp compared to the original ones. Hence, the small details are harder to distinguish
- Impact of noise: Gaussian smoothing works effectively with noise in image. For example, when noise is added in 4.2, applying Gaussian smoothing, it smooths out the noise by averaging the noised values around the neighborhood within the defined kernel size
- Image Clarity: By balancing the pixel values and reduce the noise, the image generally appears cleaner with better clarity. However, it might be traded off with resolution (less detailed)

2. Observations of Noise

- Noise Characteristics

  This is similar to what known to me as ‘white noise’. It gives the image kind of ‘rainy’ or ‘dusty’ appearance, which is similar to the old television or pictures taken by old cameras.

- Varying Noise level

  By adjusting the standard deviation, I can see the changes in noise level.

  Initially I started with relatively small values of SD (20) and the noise was subtle, not much changing in overall quality of the image, only by close inspection, some small nosie can be noticed.

  Then, by increasing the SD (around 80+), the noise became more significant and the grained textutes can be noticed with a glance, degrade the overall quality of the image.

3. Considerations

Balancing between noise reduction and preserving detail: While Gaussian Smoothing works effectively with noice reduction, overdoing it might comes with the consequence of reducing the detail level. This can be seen when applied the Gaussian Smoothing to a image with low-noise-level, while nothing changed in the noise, it downgrade the overall detail of the image.

Thus, when applied smoothing technique, it is important to consider the initial noise level of the image choose the appropriate kernel size which decide the tradeoff with detail blur.