DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
EC ENGR 188: THE FOUNDATIONS OF COMPUTER VISION

INSTRUCTOR: Prof. Achuta Kadambi
TA: Rishi Upadhyay

NAME: Simon Lee
UID: 505310387

HOMEWORK 2

PROBLEM	TYPE	TOPIC	MAX. POINTS
1	Analytical	Filter Design	10
2	Analytical + Coding	Blob Detection	10
3	Coding	Corner Detection	10
4	Analytical	2D Transformation	10
5	Analytical	Interview Question	5

Motivation

In the previous homework, we have seen how to process images using convolutions and mine images for features such as edges using image gradients. In this homework, we will detect other types of useful features in images such as blobs and corners that can be useful for a variety of computer vision tasks. We then transition from detecting useful features in images to relating different images via 2D transformations.

The problem set consists of:

- analytical questions to solidify the concepts covered in the class; and
- coding questions to implement some of the algorithms described in class using Python.

Homework Layout

The homework consists of 5 problems in total, with subparts for each problem. There are 2 types of problems in this homework - analytical and coding. We encourage you to answer all the problems using the Overleaf document; however, handwritten solutions will also be accepted. The overleaf link is here: `https://www.overleaf.com/read/ggqwqzsgmcjn#d2fb52`. The coding jupyter notebook is: `https://colab.research.google.com/drive/1Tp_8YKNiZ7HlvFE4OyT9Z84t3DzwT1ac?usp=sharing`

Submission

Submission will be done via Gradescope. You will need to submit three things: (1) this PDF with answers filled out, (2) Your .ipynb file, (3) a PDF printout of your .ipynb file with all cells executed. You do not need to create a folder, you will be able to upload all three files directly to gradescope.

1 Filter Design (10.0 points)

In the class you were taught the Central Difference/Laplacian filter for detecting the edges in an image (by computing the gradients along the horizontal and vertical directions). In the discussion we derived those filters by approximating the first (for gradient) and second derivative (for Laplacian) of a univariate function $f(x)$ using the Taylor series expansion of $f(x+h)$ and $f(x-h)$, where h is a small perturbation around x with $h = 1$ for the discrete case. These filters only consider the adjacent neighbours of the current pixel. In this question you will derive a high-order approximation for the two filters, such that the filters will use 2 adjacent neighbours. To summarize, you will be deriving a higher order approximation to the first/second derivative of $f(x)$. We will use $f^{(1)}(x), f^{(2)}(x)$ to denote the first and second derivative respectively.

1.1 Compute $f(x+h)$ (1.0 points)

Write the Taylor series approximation for $f(x+h)$ around $f(x)$, such that the approximation error tends to 0 as h^5, i.e. consider only the first 5 terms in the Taylor series approximation. Use $f^{(1)}(x)$ for the first derivative, $f^{(2)}(x)$ for the second derivative and so on.

$$f(x+h) = f(x) + f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} + \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!} + \mathcal{O}(h^5)$$

1.2 Compute $f(x-h)$ (1.0 points)

Similarly, write the Taylor series approximation for $f(x-h)$ around $f(x)$, such that the approximation error tends to 0 as h^5.

$$f(x-h) = f(x) - f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} - \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!} - \mathcal{O}(h^5)$$

1.3 Compute $f(x+h) - f(x-h)$ (1.0 points)

Using the expressions you obtained in the previous two parts, compute the expression for $f(x+h) - f(x-h)$. You may assume that $\mathcal{O}(h^5)$ tends to 0, so that you can neglect the term. You will be using this result to compute a high-order approximation to $f^1(x)$.

From the previous two answers we have:

For $f(x+h)$:

$$f(x+h) = f(x) + f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} + \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}$$

For $f(x-h)$:

$$f(x-h) = f(x) - f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} - \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}$$

Now, let's subtract $f(x-h)$ from $f(x+h)$:

$$f(x+h) - f(x-h) =$$
$$\left(f(x) + f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} + \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}\right) -$$
$$\left(f(x) - f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} - \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}\right)$$

Simplifying this, we get:

$$f(x+h) - f(x-h) = 2f^{(1)}(x)h + 2\frac{f^{(3)}(x)h^3}{3!}$$

We cna now rearrange this equation to get an approximation for $f^{(1)}(x)$

$$f^{(1)}(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(x)h^2}{3!}$$

1.4 Unknowns at each pixel (1.0 points)

Let's assume that you have access to $f(x)$, which is a 1D signal (or equivalently one row in an image). This means that you can easily obtain the values for $f(x)$, $f(x \pm h)$, $f(x \pm 2h)$ and so on (assuming appropriate zero padding for start and end values). However, for each pixel x, if you only consider using its adjacent neighbours ($\pm h$), then the equation in the previous part has 2 unknowns. What are the two unknowns? *Note:* $h = 1$ for the discrete case, so h is not an unknown. But don't substitute $h = 1$ as of now; we will substitute it later.

From the equation:

$$f^{(1)}(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(x)h^2}{3!}$$

we don;t have the $f^{(1)}(x)$ and $f^{(3)}(x)$.

1.5 Compute $f^{(1)}(x)$ (1.0 points)

From the previous part, you know that for each pixel x, if we only consider $x \pm h$, then we have 1 equation and 2 variables (underdetermined system). To mitigate this issue, we consider two adjacent neighbours for each pixel ($x \pm 2h$) in addition to $x \pm h$. Replace h with $2h$ in the previous equation and you will get another equation for that pixel. So now, for each pixel, you have 2 equations and 2 variables. One equation should have $f(x \pm h)$ and the other should have $f(x \pm 2h)$. Using these two equations, solve for $f^{(1)}(x)$.

Let's first write down the two equations. Equation 1 using $x \pm h$:

$$f^{(1)}(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(x)h^2}{3!}$$

$$2f^{(1)}(x) = \frac{f(x+h) - f(x-h)}{h} - \frac{2f^{(3)}(x)h^2}{3!} \tag{1}$$

Equation 2 using $x \pm 2h$. We can also simplify it:

$$f^{(1)}(x) = \frac{f(x+2h) - f(x-2h)}{2 \cdot 2h} - \frac{f^{(3)}(x)(2h)^2}{3!} \tag{2}$$

$$f^{(1)}(x) = \frac{f(x+2h) - f(x-2h)}{4h} - \frac{4f^{(3)}(x)h^2}{3!}$$

$$4f^{(1)}(x) = \frac{f(x+2h) - f(x-2h)}{h} - \frac{16f^{(3)}(x)h^2}{3!} \tag{3}$$

Now we can use two equations (1) and (3), and we'll solve these two equations for the two unknowns $f^{(1)}(x)$ and $f^{(3)}(x)$:

$$4f^{(1)}(x) - 2f^{(1)}(x) = \left(\frac{f(x+2h) - f(x-2h)}{h} - \frac{16f^{(3)}(x)h^2}{3!} \right) - \left(\frac{f(x+h) - f(x-h)}{h} - \frac{2f^{(3)}(x)h^2}{3!} \right)$$

However we want to cancel out the third order term so we have

$$16f^{(1)}(x) - 4f^{(1)}(x) = \left(\frac{8f(x+h) - 8f(x-h)}{h} - \frac{16f^{(3)}(x)h^2}{3!} \right) - \left(\frac{f(x+2h) - f(x-2h)}{h} - \frac{16f^{(3)}(x)h^2}{3!} \right)$$

$$12f^{(1)}(x) = \frac{8f(x+h) - 8f(x-h) - f(x+2h) - f(x-2h)}{h}$$

$$f^{(1)}(x) = \frac{1}{12}(\frac{8f(x+h) - 8f(x-h) - f(x+2h) - f(x-2h)}{h})$$

1.6 Convolution Kernel (1.0 points)

What is the convolution kernel corresponding to the new filter for $f^{(1)}(x)$? Substitute $h = 1$ for the discrete case. This filter is now a higher order central-difference filter which can be used to compute the gradients/edges.

From our derivation in section 1.5 and by plugging in $h = 1$ we should get a kernel

$$\frac{1}{12}[-1, -8, 0, 8, 1]$$

1.7 Laplacian Filter (1.0 points)

We will repeat the same exercise as the previous parts; however, now we compute a higher order approximation to the Laplacian filter. Similar to 1.3, compute the expression for $f(x+h) + f(x-h)$.

We have these two equations from part 1.3:

For $f(x+h)$:

$$f(x+h) = f(x) + f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} + \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}$$

For $f(x-h)$:

$$f(x-h) = f(x) - f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} - \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}$$

Adding $f(x+h)$ and $f(x-h)$ together:

$$f(x+h) + f(x-h) =$$
$$\left(f(x) + f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} + \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}\right) +$$
$$\left(f(x) - f^{(1)}(x)h + \frac{f^{(2)}(x)h^2}{2!} - \frac{f^{(3)}(x)h^3}{3!} + \frac{f^{(4)}(x)h^4}{4!}\right)$$

Simplify this expression:

$$f(x+h)+f(x-h)=2f(x)+2\frac{f^{(2)}(x)h^2}{2!}+2\frac{f^{(4)}(x)h^4}{4!}$$

This time the odd derivatives will cancel out and it will allow us to express the equation in terms of $f^{(2)}(x)$:

$$f^{(2)}(x)=\frac{f(x+h)+f(x-h)-2f(x)}{h^2}-\frac{f^{(4)}(x)h^2}{12}$$

1.8 Unknowns for the Laplacian (1.0 points)

Similar to 1.4, what are the two unknowns for each pixel in the expression from the previous part?

we don;t have the $f^{(2)}(x)$ and $f^{(4)}(x)$.

1.9 Compute $f^{(2)}(x)$ (1.0 points)

Similar to 1.5, use $f(x\pm 2h)$ to solve for $f^{(2)}(x)$. Write the expression for $f^{(2)}(x)$.

from 1.7 we have

$$f^{(2)}(x)=\frac{f(x+h)+f(x-h)-2f(x)}{h^2}-\frac{f^{(4)}(x)h^2}{12}$$

If we plug in $f(x\pm 2h)$

we can subsitute the equation derived from 1.7 to obtain:

$$f^{(2)}(x)=\frac{f(x+2h)+f(x-2h)-2f(x)}{(2h)^2}-\frac{f^{(4)}(x)\cdot(2h)^2}{12}+\dots$$

we rearrange and get

$$4f^{(2)}(x)=\frac{f(x+2h)+f(x-2h)-2f(x)}{h^2}-\frac{16f^{(4)}(x)\cdot h^2}{12}+\dots$$

Now we subtract the $x+h$ equation with the $x+2h$ to get:

$$16f^{(2)}(x)-4f^{(2)}(x)=\frac{16f(x+h)+16f(x-h)-32f(x)}{h^2}-(\frac{f(x+2h)+f(x-2h)-2f(x)}{h^2})$$

$$f^{(2)}(x) = \frac{1}{12}\frac{16f(x+h)+16f(x-h)-34f(x)-f(x+2h)+f(x-2h)}{h^2}$$

1.10 Convolution Kernel (1.0 points)

What is the corresponding convolution for the filter you derived in the previous part? Use $h = 1$ for the discrete case.

If we plug in $h = 1$ then we get a kernel of

$$[-1, -16, 34, 16, 1]$$

2 Blob Detection (10.0 points)

In this question, you will be using the Laplacian of Gaussian (LoG) filter to detect blobs in an image. Let's consider a 2D Gaussian $G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-(\frac{x^2+y^2}{2\sigma^2})}$. Remember that we need to smooth the image using a Gaussian filter before computing the Laplacian to prevent noise amplification. However, instead of computing the Laplacian after filtering the image with a Gaussian kernel, we can directly filter the image with the Laplacian of Gaussian filter.

2.1 Compute $\frac{\partial^2 G_\sigma(x,y)}{\partial x^2}$ (1.0 points)

Write the expression for $\frac{\partial^2 G_\sigma(x,y)}{\partial x^2}$.

We begin with the given 2D Gaussain

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We then compute the first and second partial derivtive:

$$\frac{\partial G_\sigma(x,y)}{\partial x} = -\frac{x}{\sigma^2} G_\sigma(x,y)$$

$$\frac{\partial^2 G_\sigma(x,y)}{\partial x^2} = \left(-\frac{1}{\sigma^2} + \frac{x^2}{\sigma^4}\right) G_\sigma(x,y)$$

So, the expression for $\frac{\partial^2 G_\sigma(x,y)}{\partial x^2}$ is:

$$\frac{\partial^2 G_\sigma(x,y)}{\partial x^2} = \left(-\frac{1}{\sigma^2} + \frac{x^2}{\sigma^4}\right) \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2.2 Compute $\frac{\partial^2 G_\sigma(x,y)}{\partial y^2}$ (1.0 points)

Write the expression for $\frac{\partial^2 G_\sigma(x,y)}{\partial y^2}$.

We repeat with respect to y

The first and second derivative of $G_\sigma(x,y)$ with respect to y is:

$$\frac{\partial G_\sigma(x,y)}{\partial y} = -\frac{y}{\sigma^2} G_\sigma(x,y)$$

$$\frac{\partial^2 G_\sigma(x,y)}{\partial y^2} = \left(-\frac{1}{\sigma^2} + \frac{y^2}{\sigma^4}\right) G_\sigma(x,y)$$

So, the expression for $\frac{\partial^2 G_\sigma(x,y)}{\partial y^2}$ is:

$$\frac{\partial^2 G_\sigma(x,y)}{\partial y^2} = \left(-\frac{1}{\sigma^2} + \frac{y^2}{\sigma^4}\right) \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2.3 Laplacian of a 2D Gaussian (1.0 points)

Using the results from the previous parts, write the expression for the Laplacian of a 2D Gaussian, $L(x,y)$.

The Laplacian of a function is defined as the sum of its second partial derivatives with respect to each independent variable. In the case of a 2D function, this would be:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

We can subsitute from 2.1 and 2,2 to get it expressed as $L(x,y)$::

$$L(x,y) = \left(-\frac{1}{\sigma^2} + \frac{x^2}{\sigma^4}\right) G_\sigma(x,y) + \left(-\frac{1}{\sigma^2} + \frac{y^2}{\sigma^4}\right) G_\sigma(x,y)$$

$$L(x,y) = \left(-\frac{2}{\sigma^2} + \frac{x^2+y^2}{\sigma^4}\right) G_\sigma(x,y)$$

$$L(x,y) = \left(-\frac{2}{\sigma^2} + \frac{x^2+y^2}{\sigma^4}\right) \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2.4 Scale-Normalization (1.0 points)

In class, we studied that it is important to normalize the scale of $L(x,y)$ before using it for blob detection. What is the normalization factor? Provide justification.

In this answer we derived our answer from scale space theory from this source: here

To account for scale invariance we typically normalize with σ^2. This is because the Laplacian operator's response to a blob-like structure increases with the size of the blob (which is characterized by σ in the scale-space representation).

2.5 LoG Filter (1.0 points)

(See the Jupyter notebook) Using the expression for $L(x,y)$ and the scale normalization, write a Python function which will compute the LoG Filter.

```python
def log_filter(size: int, sigma: float):
    """
    Computes the Laplacian of Gaussian (LoG) filter.

    Author: Simon Lee
    """
    try:
        # We need an odd size to have a center pixel
        if not isinstance(size, int) or size <= 0:
            raise ValueError("Size must be a positive integer.")
        size = size + 1 if size % 2 == 0 else size
        radius = size // 2

        # error handling for division by zero
        if sigma <= 0:
            raise ValueError("Sigma must be greater than zero.")

        y, x = np.ogrid[-radius:radius+1, -radius:radius+1]
        g = (1 / (2 * np.pi * sigma**2)) * np.exp(-(x**2 + y**2) / \
         (2 * sigma**2)) # Gaussain
        LoG = ((x**2 + y**2 - 2*sigma**2) / (sigma**4)) * g # LoG
        LoG_normalized = sigma**2 * LoG # Normalization

        # Ensure the sum of the filter is 0 (for edge detection property)
        LoG_normalized -= LoG_normalized.mean()

        return LoG_normalized

    except ValueError as ve:
        print(f"ValueError: {ve}")
    except Exception as e:
```

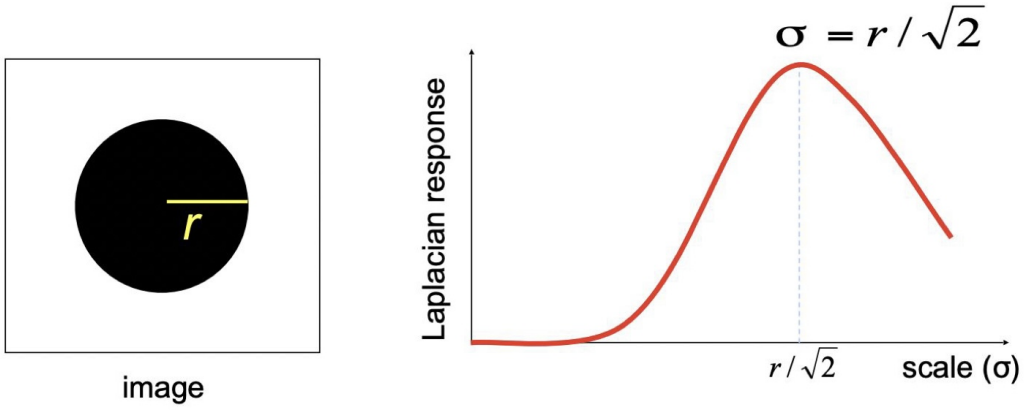
```
        print(f"An error occurred: {e}")
```

2.6 σ values (1.0 points)

(See the Jupyter notebook) What are the 5 sigma values which give the maximum response? To visualize the response, you can use the colormap in the Jupyter notebook.

Scale selection

- For a binary circle of radius r, the Laplacian achieves a maximum at



CYU @ home: can you prove why this is the case?

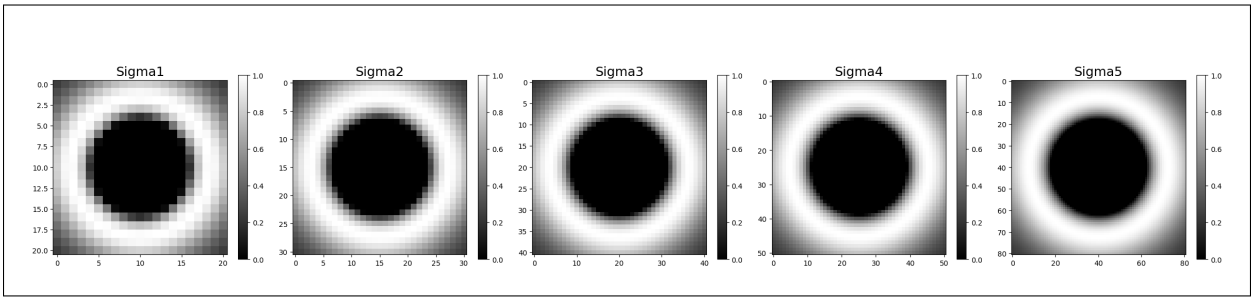
```
sigma_1 = (21/2)/math.sqrt(2)
sigma_2 =(31/2)/math.sqrt(2)
sigma_3 =(41/2)/math.sqrt(2)
sigma_4 =(51/2)/math.sqrt(2)
sigma_5 =(81/2)/math.sqrt(2)
```

However the above results ended up with a bad final product in later sections because we dont know ther radius. I therefore decided to take a brute force approach where I got better sigma values by dividing each filter by 5

```
sigma_1 = 21/5
sigma_2 =31/5
sigma_3 =41/5
sigma_4 =51/5
sigma_5 =81/5
```

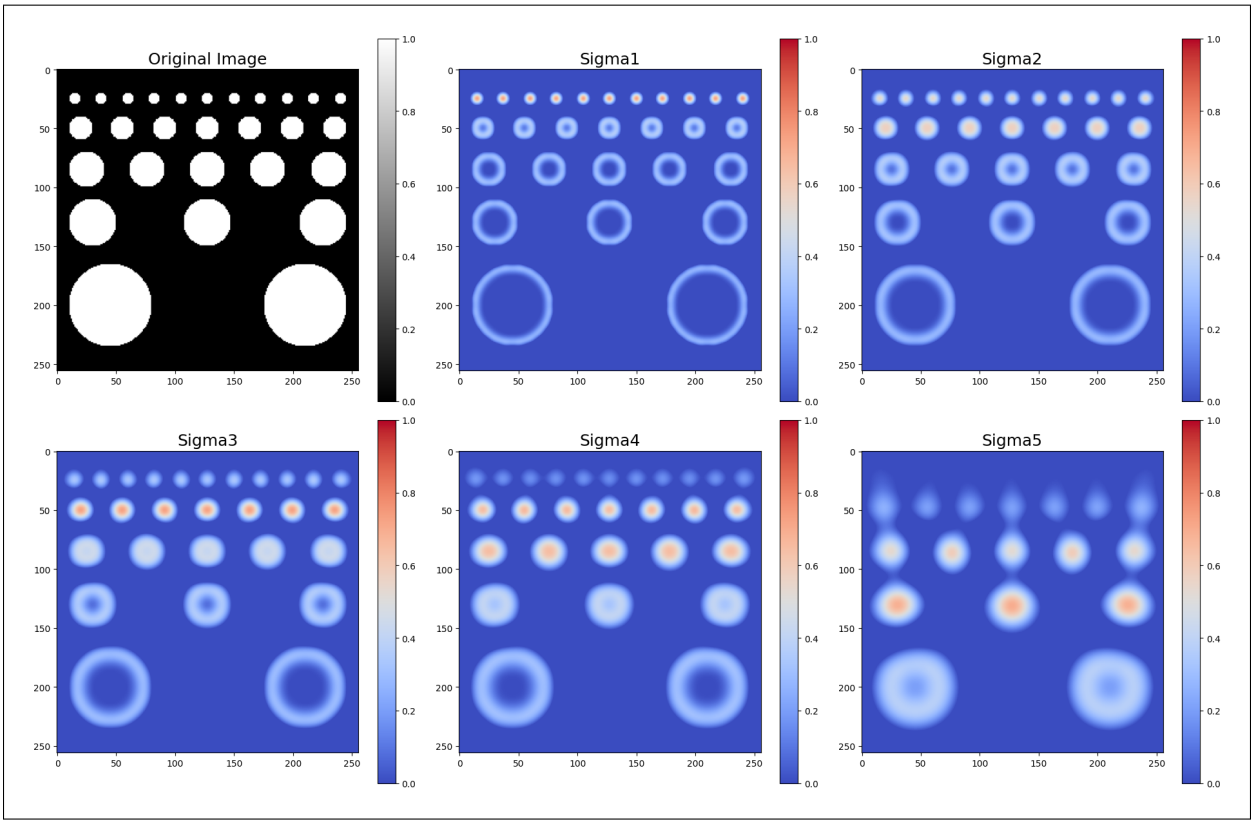
2.7 Visualize LoG Filter (1.0 points)

(See the Jupyter notebook) In this sub-part you will visualize the LoG filter. Copy the saved image from the Jupyter notebook here.



2.8 Visualize the blob detection results (3.0 points)

(See the Jupyter notebook) In this sub-part you will visualize the blob detection results. Copy the saved image from the Jupyter notebook here.


Original Image
Sigma1
Sigma2
Sigma3
Sigma4
Sigma5

3 Corner Detection (10.0 points)

In this question, you will be implementing the Harris corner detector. As discussed in class, corners generally serve as useful features.

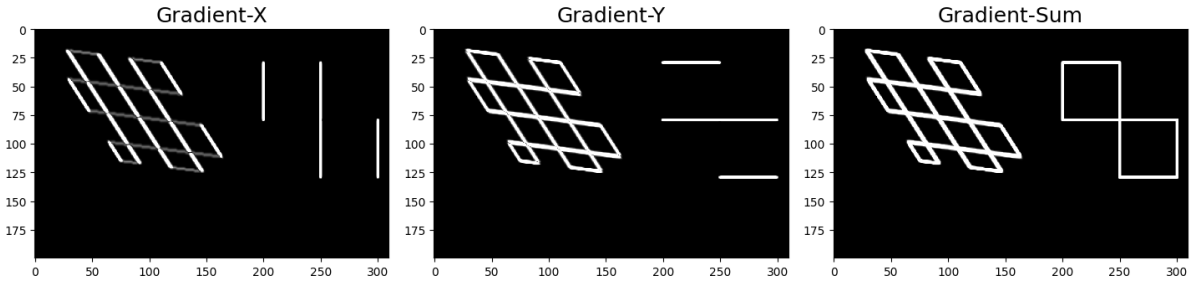
3.1 Computing Image Gradients Using Sobel Filter (1.0 points)

(See the Jupyter notebook). In this sub-part, you will write a function that computes image gradients using the Sobel filter. Make sure that your code is within the bounding box.

```
def compute_image_gradient(image: np.array):
    """
    Compute the horizontal and vertical image gradients using

    Sobel filters.

    Author Simon Lee
    """
    gradient_x = conv2D(image, sobel_x)
    gradient_y = conv2D(image, sobel_y)

    return gradient_x, gradient_y
```

3.2 Visualizing the Image Gradients (1.0 points)

(See the Jupyter notebook). In this sub-part, you will visualize the image gradients. Copy the saved image from the Jupyter notebook here.



3.3 Computing the Covariance Matrix (1.0 points)

(See the Jupyter notebook). In this sub-part, you will write a function that computes the covariance matrix of the image gradients. Make sure that your code is within the bounding box.

```python
def grad_covariance(image: np.array, size: int):
    """
    Computes the covariance matrix

    Author Simon Lee
    """
    gradient_x, gradient_y = compute_image_gradient(image)

    # Compute the squares and product of the gradients
    Ixx = gradient_x ** 2
    Iyy = gradient_y ** 2
    Ixy = gradient_x * gradient_y
    avg_filter = average_filter(size) # make avergae filter to convolve

    # Convolve each matrix with the average filter
    Ixx = conv2D(Ixx, avg_filter)
    Iyy = conv2D(Iyy, avg_filter)
    Ixy = conv2D(Ixy, avg_filter)

    return Ixx, Ixy, Iyy
```

3.4 Harris Corner Response (1.0 points)

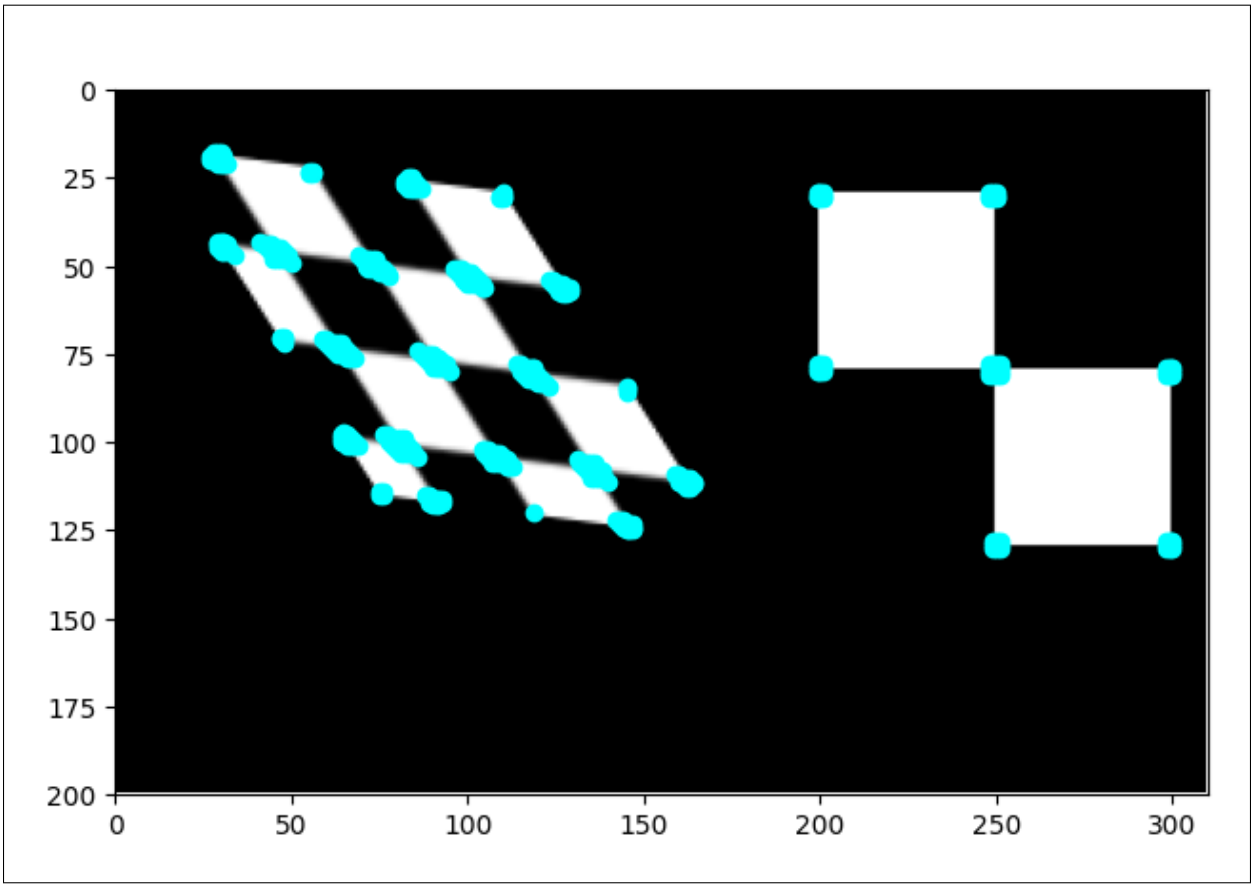
(See the Jupyter notebook). In this sub-part, you will write a function that computes the Harris response function. Make sure that your code is within the bounding box.

```python
def harris_response(image: np.array, k: float, size: int):
    """
    Computes the Harris response function for each pixel in the image.

    Author: Simon Lee
    """
    try:
        # Compute the covariance matrix of gradients
        Ixx, Ixy, Iyy = grad_covariance(image, size)
    except ValueError as e:
        raise ValueError(f"Error computing gradient covariance: {e}")

    try:
        # Compute the Harris response for each pixel
        Det_M = Ixx * Iyy - Ixy ** 2
```

```
        Trace_M = Ixx + Iyy
        R = Det_M - k * (Trace_M ** 2)
    except Exception as e:
        raise ValueError(f"Error computing Harris response: {e}")

    return R
```

3.5 Visualizing the Harris Corner Detector (1.0 points)

(See the Jupyter notebook). In this sub-part, you will visualize the Harris corner detections. Copy the saved image from the Jupyter notebook here.



3.6 Thresholding the Harris Response (1.0 points)

To remove duplicate detections, you will write a function that applies non-maximum suppression to the Harris corner detections. To make writing this function easier, you will implement it in various parts.

(See the Jupyter notebook). In this sub-part, you will implement the first step of non-maximum

suppression: thresholding the Harris response to obtain candidate corner detections. Make sure that your code is within the bounding box.

```
def threshold_harris_response(harris_response: np.array, threshold:
↪ float):
    """
    Thresholds the Harris response map to obtain pixel locations with
    response greater than the threshold.

    Author Simon Lee
    """
    # Find indices where the response is greater than the threshold
    indices = np.argwhere(harris_response > threshold)

    return indices
```

3.7 Sorting Candidate Detections (1.0 points)

(See the Jupyter notebook). In this sub-part, you will sort the candidate detections by maximum Harris response value. Make sure that your code is within the bounding box.

```
def sort_detections(candidate_detections: np.array, harris_response:
↪ np.array):
    """
    Sorts the candidate detections based on the Harris response value
    ↪ in descending order.

    Author: Simon Lee
    """
    # Extract the Harris response values for each candidate detection
    # and sort in descending order
    sorted_indices = np.argsort([harris_response[i, j] for i, j in
    ↪ candidate_detections])[::-1]

    # Arrange the detections in sorted order
    sorted_detections = candidate_detections[sorted_indices]

    return sorted_detections
```

3.8 Suppressing Non-max Detections (1.0 points)

(See the Jupyter notebook). In this sub-part, you will implement the final step of non-maximum suppression: removing corner detections that are not local maxima. Make sure that your code is within the bounding box.

```
def local_max(sorted_detections: np.array, distance: float):
    """
    Returns the detections that are local maxima.

    Author: Simon Lee
    """
    local_maxima = []
    for current_detection in sorted_detections:
        # Compare current detection to all detections that are already
        ↪ considered as local maxima
        if all(l2_distance(current_detection, existing_detection) >
        ↪ distance for existing_detection in local_maxima):
            local_maxima.append(current_detection)

    return np.array(local_maxima)
```

3.9 Non-Maximum Suppression: Putting it all together (1.0 points)

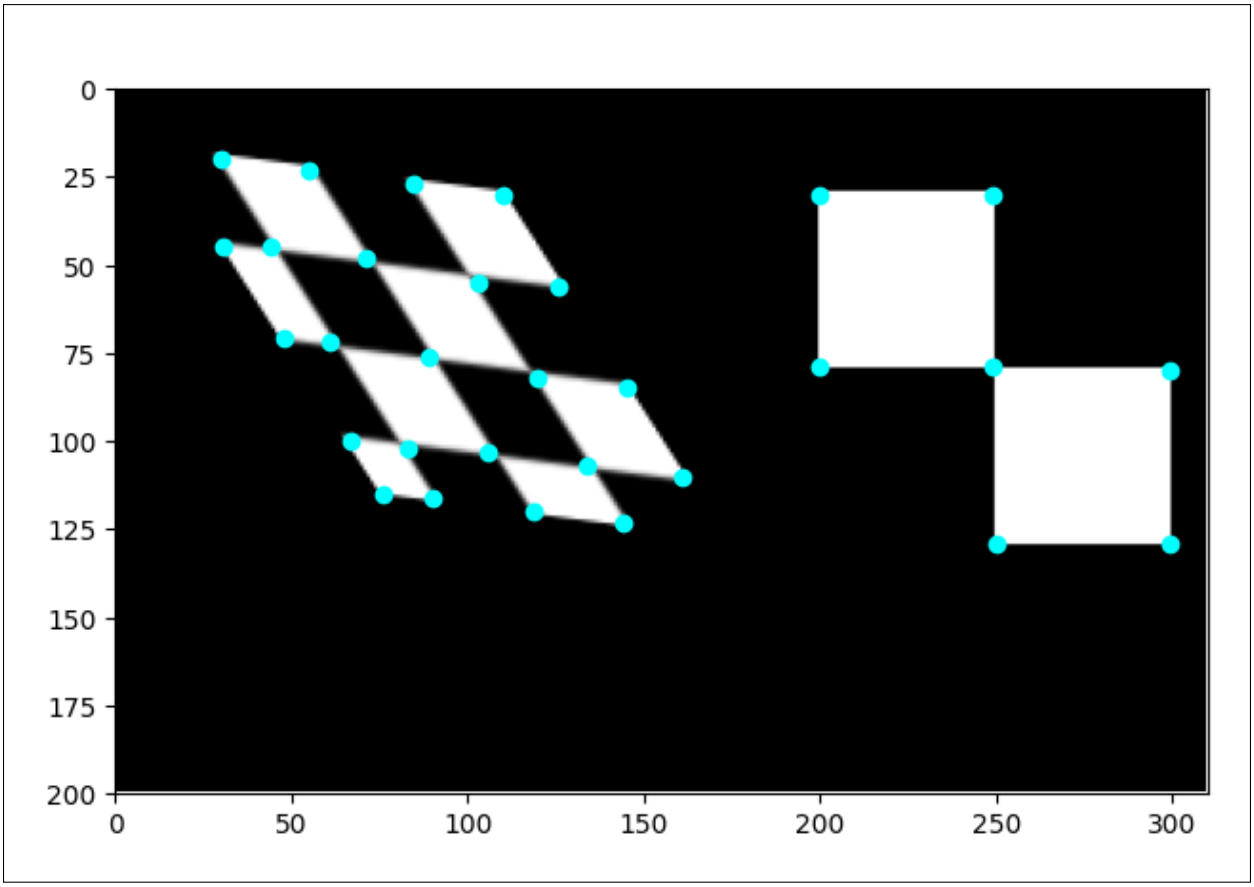
(See the Jupyter notebook). In this sub-part, you will write a function that performs non-maximum suppression on the Harris corner response. Make sure that your code is within the bounding box.

```
def non_max_suppression(harris_response: np.array, distance: float,
↪ threshold: float):
    """
    Applies non-maximum suppression to the Harris corner response to
    ↪ get corner detections.
    using 3 previous functions we wrote

    Author: Simon Lee
    """
    # run all three functions
    candidate_detections = threshold_harris_response(harris_response,
    ↪ threshold)
    sorted_detections = sort_detections(candidate_detections,
    ↪ harris_response)
    corners = local_max(sorted_detections, distance)
```

```
    return corners
```

3.10 Visualizing Harris Corner Detections + Non-maximum Suppression (1.0 points)

(See the Jupyter notebook). In this sub-part, you will visualize the Harris corner detections after non-maximum suppression has been applied. Copy the saved image from the Jupyter notebook here. Duplicate corner detections should now be removed.



4 2D Transformation (10.0 points)

In this question, you will be identifying different 2D transformations. You will be given a set of feature points x and the corresponding transformed points x'. Given these two set of points, you have to identify the 2D transformation. For the first 5 sub-parts, there is only one transformation (translation, scaling, rotation, shearing). For the next parts, there may be more than one transformation. While justifying your answer for each part you should also write the 3×3 transformation matrix M.

4.1 Example 1 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(2,2),(3,2),(3,3),(2,3)\}$. Identify the transformation and justify:

each point in x is translated by $(+1,+1)$ to get x', the transformation is a **translation**.

The general form of a 2D transformation matrix for translation is:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

For this specific case, $t_x = +1$ and $t_y = +1$, so the transformation matrix M is:

$$M = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

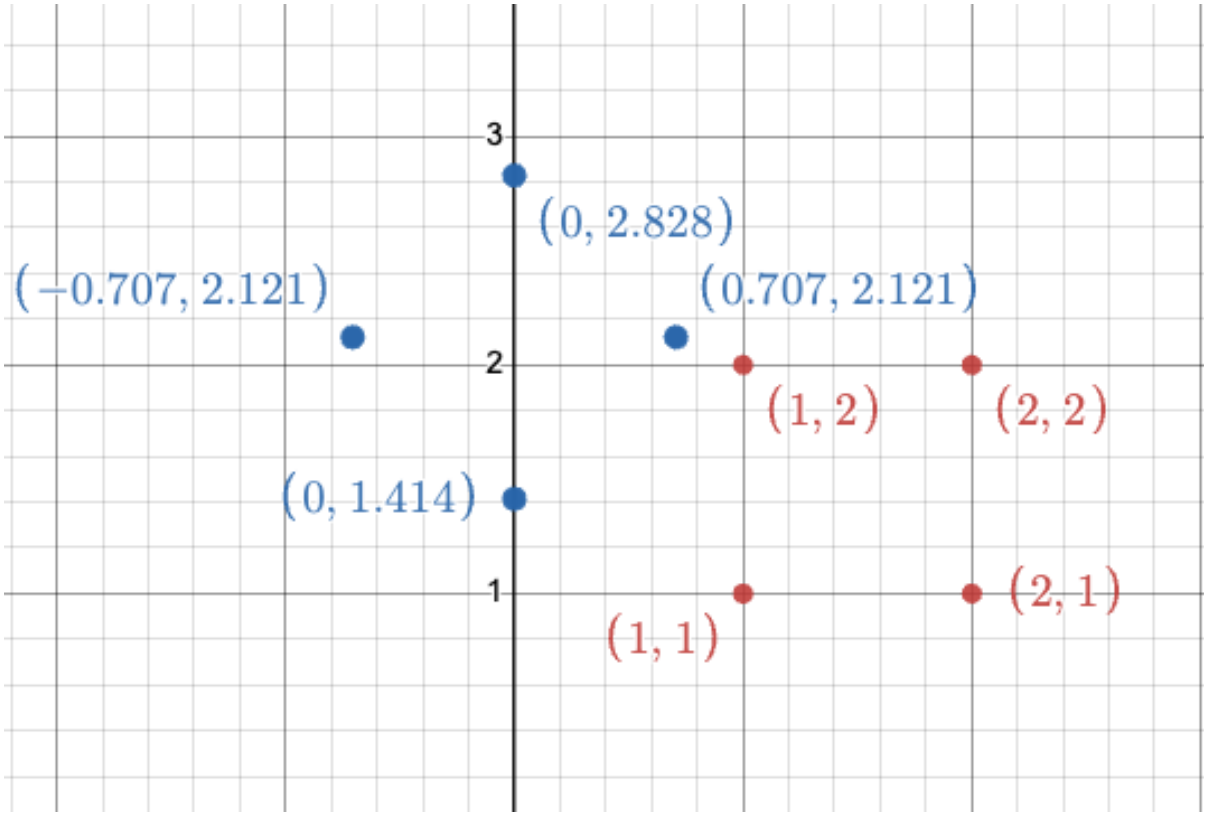
This matrix confirms that the transformation is a translation by $(+1,+1)$.

4.2 Example 2 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(0,\sqrt{2}),(\sqrt{2}-\frac{1}{\sqrt{2}},\sqrt{2}+\frac{1}{\sqrt{2}}),(0,2\sqrt{2}),(\frac{1}{\sqrt{2}}-\sqrt{2},\sqrt{2}+\frac{1}{\sqrt{2}})\}$. Identify the transformation and justify:

We begin the problem by simplifying the readout $x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(0,\sqrt{2}),(\frac{1}{\sqrt{2}},\frac{3}{\sqrt{2}}),(0,2\sqrt{2}),(\frac{-1}{\sqrt{2}},\frac{3}{\sqrt{2}})\}$.

I aslo graphed it to see the transformation:



Given the original and transformed points and the graph, the transformation can be identified as a rotation.

Each point in x is rotated by 45 degrees to get x'. This observation is based on the analysis that the points after transformation are neither uniformly translated, scaled, nor sheared, but their positions indicate a rotation around a point, maintaining the same relative distances among them, which aligns with the properties of a rotation.

The general form of a 2D transformation matrix for rotation by an angle θ is:

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For this specific case, where $\theta = 45^\circ$ or $\theta = \frac{\pi}{4}$ radians, the transformation matrix M is:

$$M = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.3 Example 3 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(-1,1),(-1,2),(-2,2),(-2,1)\}$. Identify the transformation and justify:

it appears to be a reflection across the y-axis. In this case, each point (x,y) transforms to $(-x,y)$

The general form of a 2D transformation matrix for reflection across the y-axis is:

$$M = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.4 Example 4 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(3,5),(6,5),(6,10),(3,10)\}$. Identify the transformation and justify:

This is an example of scaling

The general form of a 2D transformation matrix for scaling is:

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

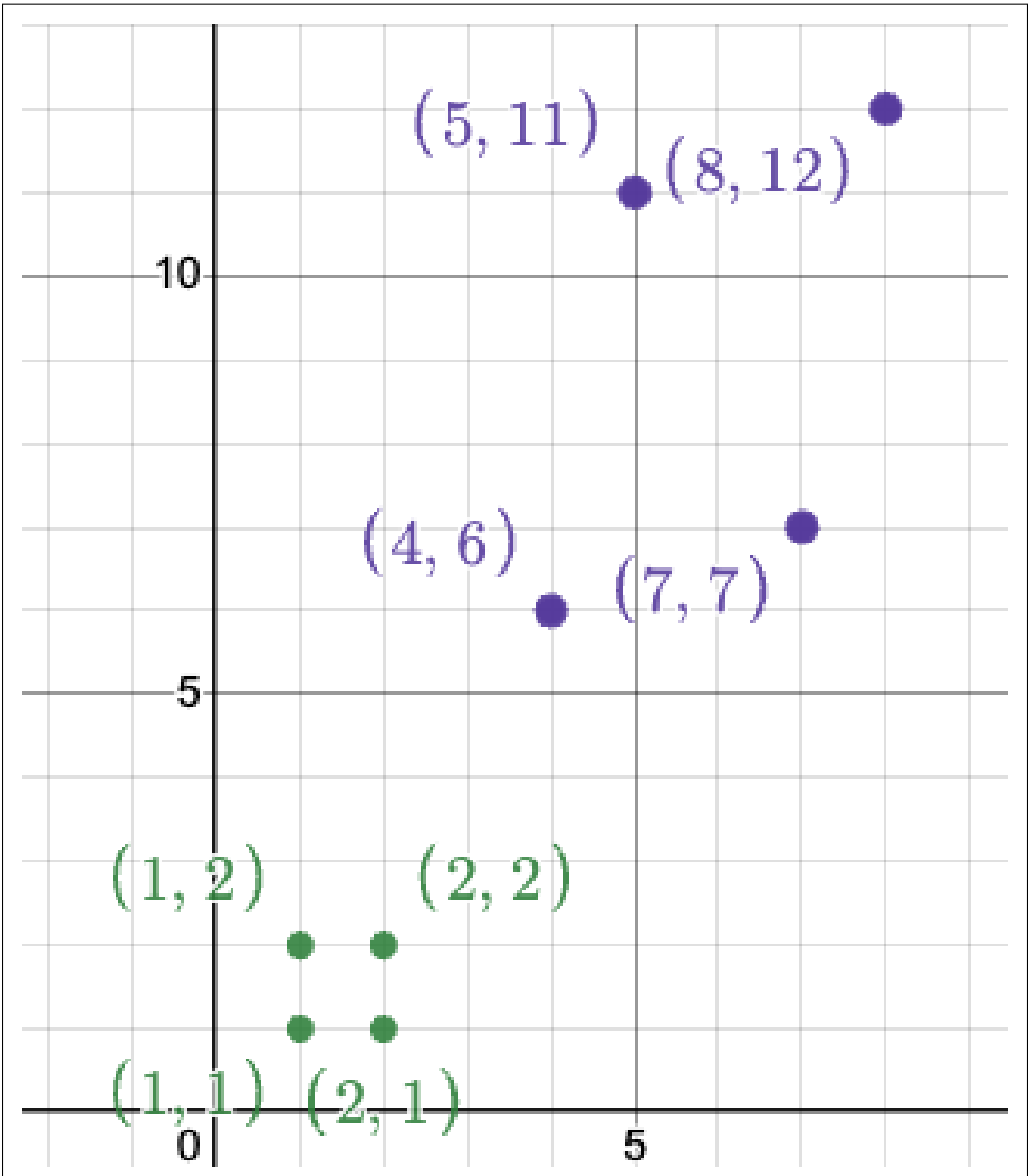
For this specific case, the scaling factors are $s_x = 3$ and $s_y = 5$, so the transformation matrix M is:

$$M = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.5 Example 5 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(4,6),(5,11),(8,12),(7,7)\}$. Identify the transformation and justify:

Because the transformation is not clear, I again graph the points:



This is an example of shearing.

The general form of a 2D transformation matrix that incorporates shearing and scaling is:

$$M = \begin{bmatrix} 1 & \text{sh}_x & 0 \\ \text{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For this specific case, given the solution from the equations, the transformation matrix M is identified as primarily representing shearing, with the shearing factors being $\text{sh}_x = 3$ and $\text{sh}_y = 5$. Thus, the transformation matrix M is:

$$M = \begin{bmatrix} 1 & 3 & 0 \\ 5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.6 Example 6 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(0,2),(0,3),(-1,3),(-1,2)\}$. Identify the two transformations and their order and justify:

There is a rotation and a translation involved.

The transdormations involved are a 90 degree rotation as indicated by the -1 in the x direction and 1 in the y direction values

$$M_{rotation} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and a translation of 1 and 1 in both x and y direction:

$$M_{translation} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The combined transformation matrix M is obtained by multiplying these two matrices:

$$M = M_{rotation} \times M_{translation}$$

$$M_{combined} = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

4.7 Example 7 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(-2,2),(-2,3),(-3,3),(-3,2)\}$. Identify the two transformations and their order and justify:

There is again a 90 degree rotation and a translation involved.

The transdormations involved are a 90 degree rotation shows by the -1 in the x direction and 1 in the y direction values

$$M_{eorrion} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and a translation of -1 and 1 in both x and y direction:

$$M_{translation} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The combined transformation matrix M is obtained by multiplying these two matrices in this order:

$$M = M_{rotation} \times M_{translation}$$

$$M_{combined} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

4.8 Example 8 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(4,6),(7,6),(7,11),(4,11)\}$. Identify the two transformations and their order and justify:

There is a scaling and a translation involved.

Scaling by a factor of 3 in the x-direction and a factor of 5 in the y-direction:

$$M_{scaling} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation 3 units to the right and 5 units up:

$$M_{translation} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The combined transformation matrix M is obtained by multiplying these two matrices:

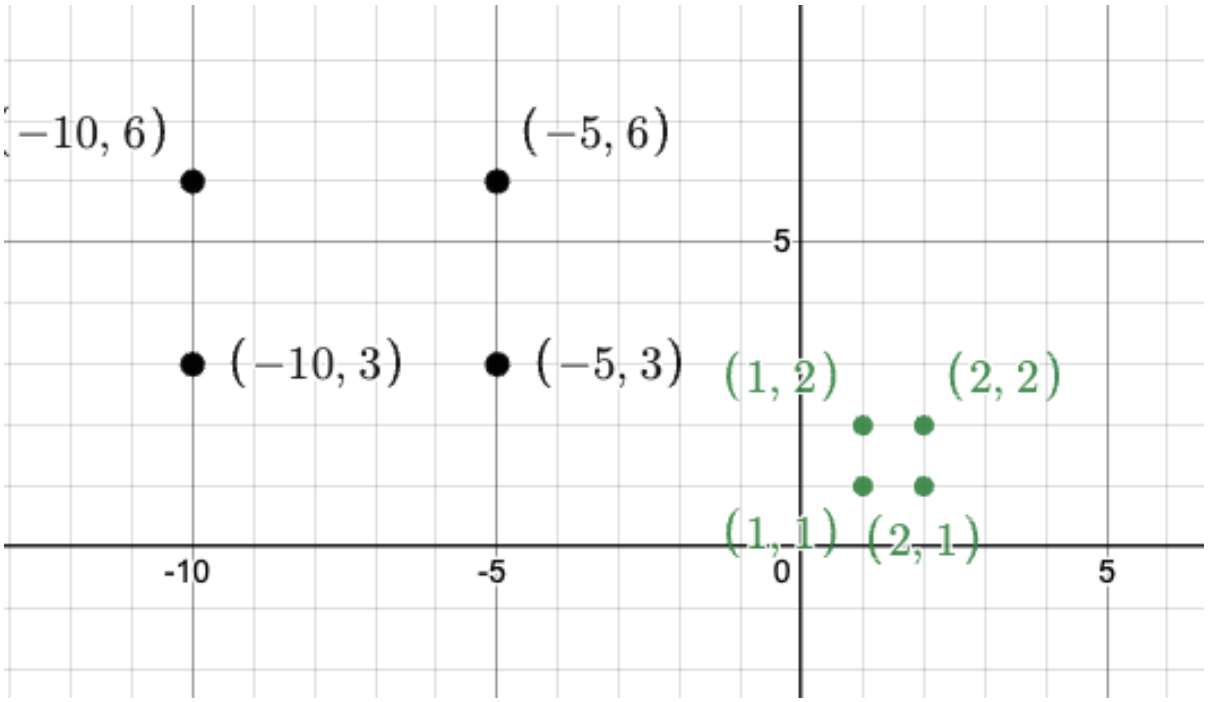
$$M = M_{translation} \times M_{scaling}$$

$$M = \begin{bmatrix} 3 & 0 & 1 \\ 0 & 5 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

4.9 Example 9 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(-5,3),(-5,6),(-10,6),(-10,3)\}$. Identify the two transformations and their order and justify:

This one is a little tricky so I included the graph



There is a reflection on the y axis and a scaling involved

Scaling by a factor of 3 in the x-direction and a factor of 5 in the y-direction:

$$M_{scaling} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation 3 units to the right and 5 units up:

$$M_{translation} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

The combined transformation matrix M is obtained by multiplying these two matrices:

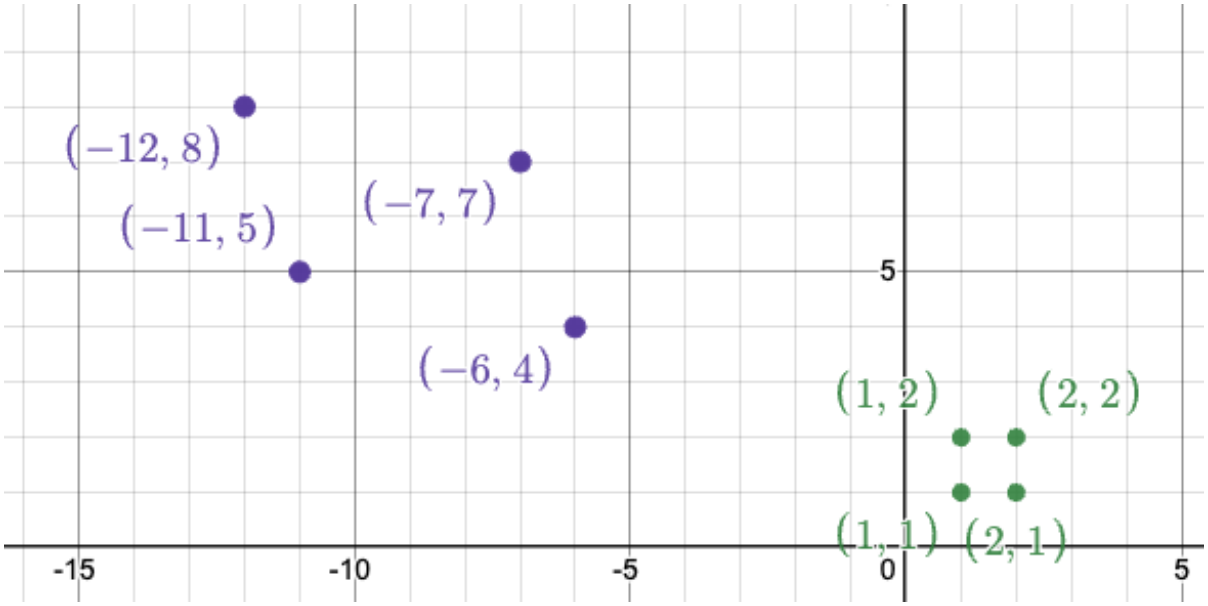
$$M = M_{translation} \times M_{scaling}$$

$$M = \begin{bmatrix} 0 & -5 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.10 Example 10 (1.0 points)

$x = \{(1,1),(2,1),(2,2),(1,2)\}$ and $x' = \{(-6,4),(-11,5),(-12,8),(-7,7)\}$. Identify the two transformations and their order and justify:

We graph the transformation to understand what it going on



The two transformations involved in this example are a shearing and a scaling.

Scaling is first applied:

$$M_{reflection+scaling} \begin{bmatrix} -5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Next we shear -1 along the x axis and 1 on the y axis:

$$M_{translation} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The combined transformation matrix M is obtained by multiplying these two matrices:

$$M = M_{translation} \times M_{reflection+scaling}$$

$$M = \begin{bmatrix} -5 & -1 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5 Interview Question (5.0 points)

Object detection is a technique which allows computers to identify and localize objects in images/videos. Let us consider that we have an object detection system that finds cars in a given image. The object detection system will propose some candidates for the object. You can see multiple candidates for the car in Figure 1 (left), and the final bounding box for the car in Figure 1 (right). Each bounding box has a score which measures the likelihood of finding a car. Given the set of bounding boxes with their locations and their scores, propose a method for generating just one bounding box per object. Use one of the algorithms that has been covered in the class or even this homework.

Hint: One metric for measuring whether different bounding boxes are in the same local "neighborhood" is intersection over union (IoU), which is defined as follows:

$$\text{IoU}(\text{box1}, \text{box2}) = \frac{\text{Area of intersection of box1 and box2}}{\text{Area of union of box1 and box2}}.$$

Figure 1: (Left) The object detection system identifies many candidates for the location of the car. For each candidate, there is a score which measures how likely it is to find a car in that bounding box. You can see that there are multiple red boxes, where each box will have a score between 0-1. (Right) The final bounding box for the car.

Just like we just did in part 3 for reducing the redundancy of the harris corner detection, we could probably use Non-Maximum Suppression to better localize and obtain an image like we see on the right. So like problem 3 we can follow a similar appraoch. We can then get a group

of bounding boxes, sort them by their IOU metric as given by the problem then apply non maximum suprression and return the output.