Department of Electrical and Computer Engineering, UCLA
EC ENGR 188: The Foundations of Computer Vision

Instructor: Prof. Achuta Kadambi
TA: Howard Zhang, Shijie Zhou

Name: Haoxuan Li
UID: 705738656

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.

Submission

You will need to make two submissions: (1) Gradescope: You will submit a PDF with all the answers on Gradescope. (2) BruinLearn: You will submit your Jupyter notebook (.ipynb file) with filename {your UID}.ipynb with all the cells executed on BruinLearn.

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.

$$f(x+h) = f(x) + hf^{(1)}(x) + \tfrac{h^2}{2}f^{(2)}(x) + \tfrac{h^3}{6}f^{(3)}(x) + \tfrac{h^4}{24}f^{(4)}(x) + \mathcal{O}(h^5)$$

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.

$$f(x-h) = f(x) - hf^{(1)}(x) + \tfrac{h^2}{2}f^{(2)}(x) - \tfrac{h^3}{6}f^{(3)}(x) + \tfrac{h^4}{24}f^{(4)}(x) - \tfrac{h^5}{120}f^{(5)}(x) + \mathcal{O}(h^6))$$

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 $f^1(x)$.

$$f(x+h) - f(x-h) = 2hf^{(1)}(x) + \tfrac{h^3}{3}f^{(3)}(x) + \tfrac{h^5}{60}f^{(5)}(x) + \mathcal{O}(h^6)$$

Assuming $\mathcal{O}(h^5)$ tends to 0, we can approximate $f(x+h) - f(x-h) \approx 2hf^{(1)}(x)$

Therefore high-order approximation of $f^1(x) \approx \frac{f(x+h)-f(x-h)}{2h}$

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.

1. $f^{(1)}(x)$, the value of the first derivative of the signal at pixel x.
2. $f^{(3)}(x)$, the value of the third derivative of the signal at pixel 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 $f^{(1)}(x)$.

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

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.

$$f(x \pm 1) = 2f^{(1)}(x) + \mathcal{O}(h^4) \rightarrow f^{(1)}(x) = \tfrac{1}{2}(f(x \pm 1)) + \mathcal{O}(h^4)$$
$$f(x \pm 2) = 4f^{(1)}(x) + \mathcal{O}(h^4) \rightarrow f^{(1)}(x) = \tfrac{1}{4}(f(x \pm 2)) + \mathcal{O}(h^4)$$

Either $[\frac{1}{2}, 0, -\frac{1}{2}]$ or $[\frac{1}{4}, 0, -\frac{1}{4}]$

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 $f(x+h)+f(x-h)$.

$f(x+h) = f(x) + hf^{(1)}(x) + \frac{h^2}{2}f^{(2)}(x) + \frac{h^3}{6}f^{(3)}(x) + \frac{h^4}{24}f^{(4)}(x) + \frac{h^5}{120}f^{(5)}(x) + \mathcal{O}(h^6)$

$f(x-h) = f(x) - hf^{(1)}(x) + \frac{h^2}{2}f^{(2)}(x) - \frac{h^3}{6}f^{(3)}(x) + \frac{h^4}{24}f^{(4)}(x) - \frac{h^5}{120}f^{(5)}(x) + \mathcal{O}(h^6)$

$f(x+h)+f(x-h) = 2f(x)+h^2f^{(2)}(x)+\frac{2h^4}{24}f^{(4)}(x)+\mathcal{O}(h^6) \ = 2f(x)+h^2f^{(2)}(x)+\frac{h^4}{12}f^{(4)}(x)+\mathcal{O}(h^6)$

Neglecting higher order terms $f(x+h)+f(x-h) \approx 2f(x)+h^2f^{(2)}(x)$

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?

1. $f^{(2)}(x)$, the value of the second derivative of the signal at pixel x.
2. $f^{(4)}(x)$, the value of the fourth derivative of the signal at pixel 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 $f^{(2)}(x)$.

$f^{(2)}(x) \approx \frac{f(x+h)+f(x-h)-2f(x)}{h^2}$

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.

Assuming $h = 1$ for the discrete case, $f^{(2)}(x) \approx f(x+1)+f(x-1)-2f(x)$

Therefore the corresponding convolution kernel for Lapacian filter is $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

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

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

$$\frac{\partial^2 G_\sigma(x,y)}{\partial x^2} = \frac{\partial G_\sigma(x,y)}{\partial x}\left(-\tfrac{x}{\pi\sigma^4}e^{-(\frac{x^2+y^2}{2\sigma^2})}\right) = -\tfrac{1}{\pi\sigma^4}e^{-(\frac{x^2+y^2}{2\sigma^2})} + \tfrac{x^2}{\pi\sigma^6}e^{-(\frac{x^2+y^2}{2\sigma^2})}$$

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

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

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

$$\frac{\partial^2 G_\sigma(x,y)}{\partial y^2} = \frac{\partial G_\sigma(x,y)}{\partial y}\left(-\tfrac{y}{\pi\sigma^4}e^{-(\frac{x^2+y^2}{2\sigma^2})}\right) = -\tfrac{1}{\pi\sigma^4}e^{-(\frac{x^2+y^2}{2\sigma^2})} + \tfrac{y^2}{\pi\sigma^6}e^{-(\frac{x^2+y^2}{2\sigma^2})}$$

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, $L(x,y)$.

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

$$\nabla^2 G_\sigma(x,y) = -\frac{1}{\pi\sigma^4} e^{-\left(\dfrac{x^2+y^2}{2\sigma^2}\right)} + \frac{x^2}{\pi\sigma^6} e^{-\left(\dfrac{x^2+y^2}{2\sigma^2}\right)} + -\frac{1}{\pi\sigma^4} e^{-\left(\dfrac{x^2+y^2}{2\sigma^2}\right)} + \frac{y^2}{\pi\sigma^6} e^{-\left(\dfrac{x^2+y^2}{2\sigma^2}\right)}$$

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

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.

The normalization factor is σ^2. This ensures that the Laplacian is normalized with respect to scale. Without it, value of $\nabla^2 G_\sigma(x,y)$ would increase with increasing values of σ for $\frac{1}{\sigma^2}\nabla^2 G_\sigma(x,y)$

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.

```
# Write your answer in this cell.
def log_filter(size: int, sigma: float):
    # Create a 2D Gaussian kernel
    kernel = np.zeros((size, size))
    for i in range(size):
        for j in range(size):
            kernel[i][j] = -1 * np.exp(-((i-size//2)**2 + (j-size//2)**2)
            /(2*sigma**2)) * (1-((i-size//2)**2 + (j-size//2)**2)
            /(2*sigma**2)) / (np.pi * sigma**2)

    return kernel
```

2.6 σ values (1.0 points)

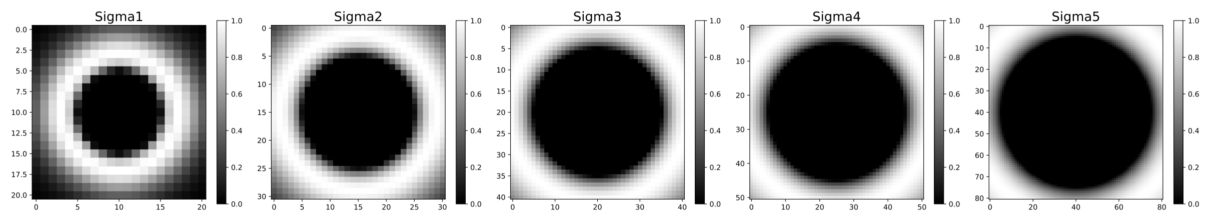
(See the Jupyter notebook) What are the 5 sigma values which give the maximum response? To visualize the response, you can use the colormap in the Jupyter notebook.

```
# sigma = r / sqrt(2)
sigma_1 = 3.5
```

```
log_1 = log_filter(21, sigma_1)
sigma_2 = 7.0
log_2 = log_filter(31, sigma_2)
sigma_3 = 10.6
log_3 = log_filter(41, sigma_3)
sigma_4 = 14.1
log_4 = log_filter(51, sigma_4)
sigma_5 = 24.7
log_5 = log_filter(81, sigma_5)
```

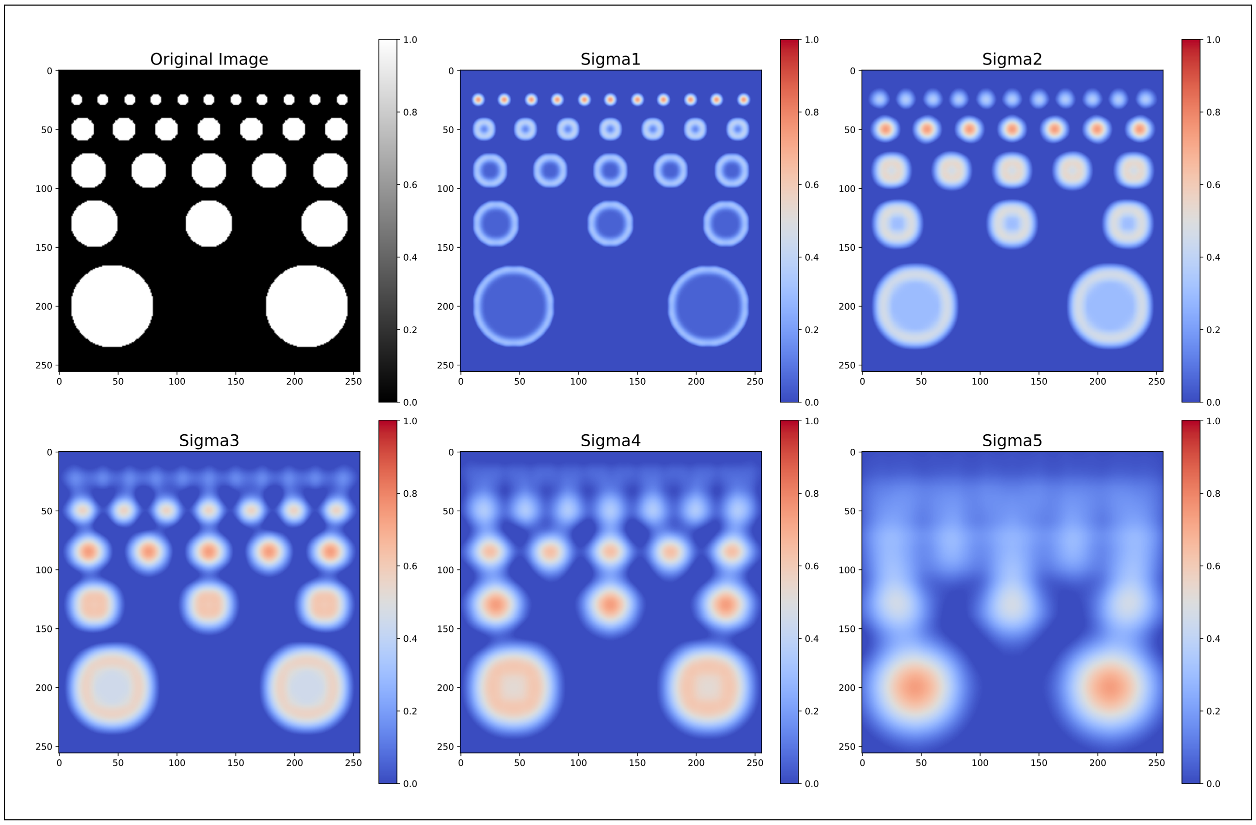
2.7 Visualize LoG Filter (1.0 points)

(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)

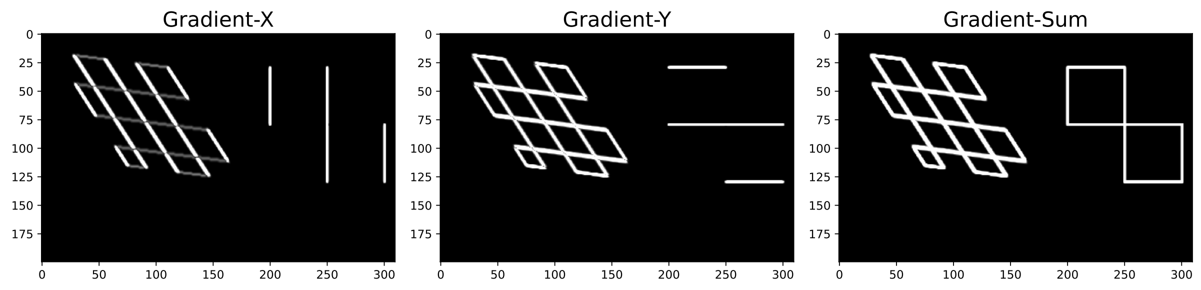
(See the Jupyter notebook). In this sub-part, you will write a function that computes image gradients using the Sobel filter. Make sure that your code is within the bounding box.

```
def compute_image_gradient(image: np.array):
  grad_x = conv2D(image, sobel_x)
  grad_y = conv2D(image, sobel_y)

  return grad_x, grad_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.

```
def grad_covariance(image: np.array, size: int):
  avg_filter = average_filter(size)
  I_x = conv2D(image, np.array([[-1,0,1],[-1,0,1],[-1,0,1]]))
  I_y = conv2D(image, np.array([[-1,-1,-1],[0,0,0],[1,1,1]]))
  I_xx = conv2D(np.square(I_x), avg_filter)
  I_yy = conv2D(np.square(I_y), avg_filter)
  I_xy = conv2D(I_x*I_y, avg_filter)
  return I_xx, I_xy, I_yy
```

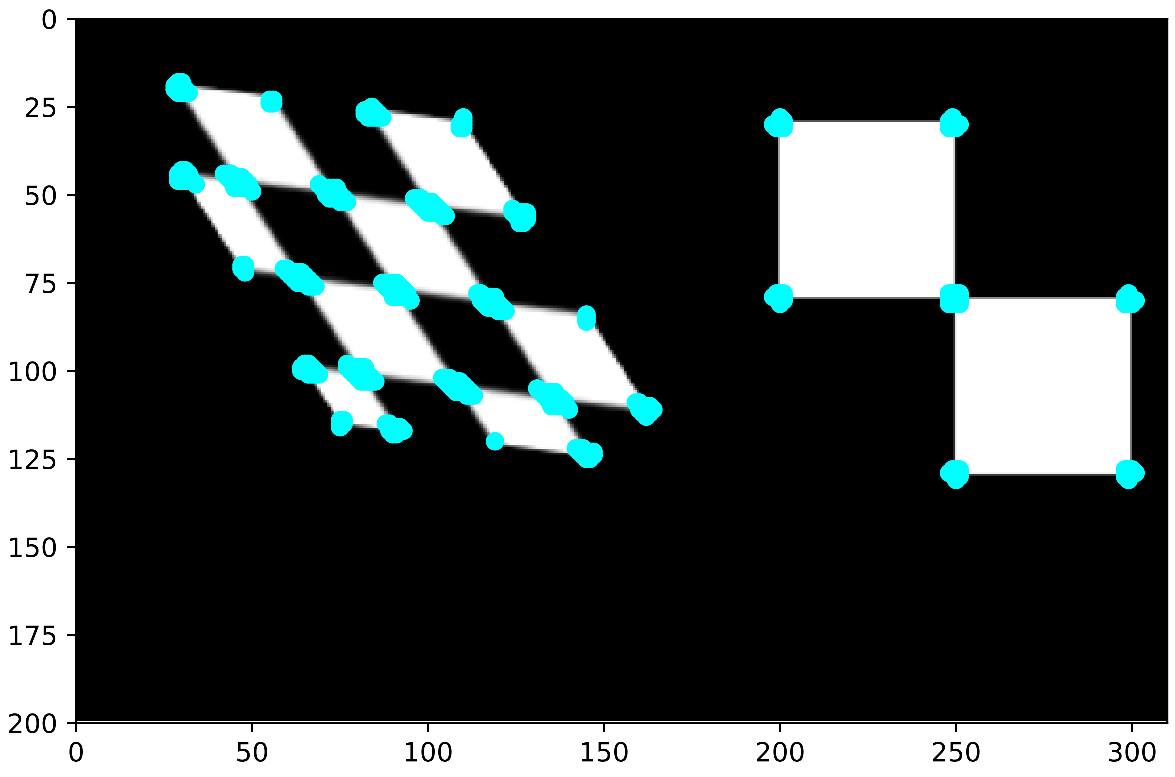
3.4 Harris Corner Response (1.0 points)

(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.

```
def harris_response(image: np.array, k: float, size: int):
  I_xx, I_xy, I_yy = grad_covariance(image, size)
  det = (I_xx*I_yy) - (I_xy**2)
  trace = I_xx + I_yy
  R = det - k*(trace**2)
  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):
    candidate_detections = np.argwhere(harris_response > threshold)

    return candidate_detections
```

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):
    sorted_detections = candidate_detections[np.argsort(harris_response[
    candidate_detections[:, 0], candidate_detections[:, 1]])[::-1]]

    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):
    remaining_detections = []
    for detection in sorted_detections:
        if len(remaining_detections) == 0:
            # Add the first detection to the remaining detections
            remaining_detections.append(detection)
        else:
            # Check if the current detection is within distance of any
             ↪  of the remaining detections
            is_local_max = True
            for remaining_detection in remaining_detections:
                if l2_distance(detection, remaining_detection) <
                 ↪  distance:
                    is_local_max = False
                    break
            # If the current detection is not within distance of any of
             ↪  the remaining detections, add it to the remaining
             ↪  detections
            if is_local_max:
                remaining_detections.append(detection)

    return np.array(remaining_detections)
```

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

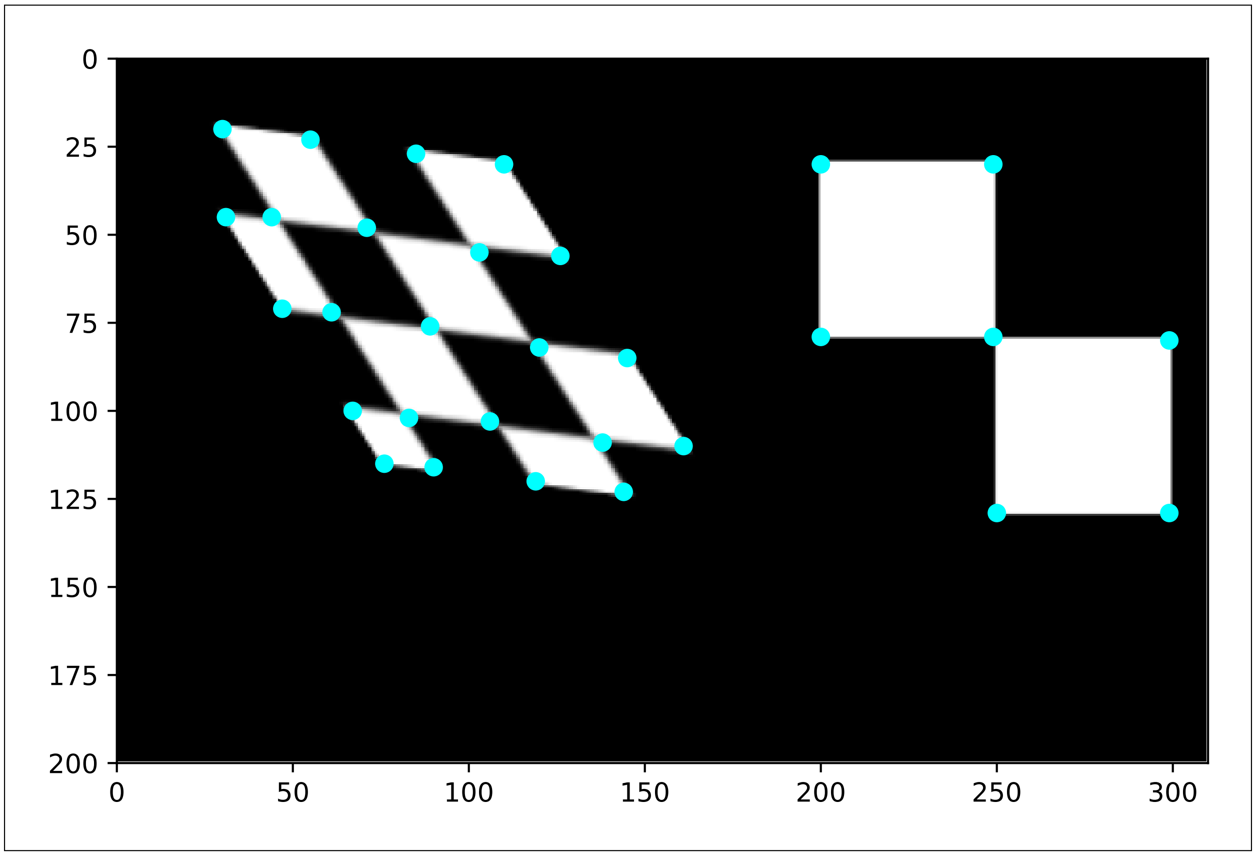
(See the Jupyter notebook). In this sub-part, you will write a function that performs non-maximum suppression on the Harris corner response. Make sure that your code is within the bounding box.

```
def non_max_suppression(harris_response: np.array, distance: float,
 ↪  threshold: float):
    candidate_detections = threshold_harris_response(harris_response,
     ↪  threshold)
    sorted_detections = sort_detections(candidate_detections,
     ↪  harris_response)
    remaining_detections = local_max(sorted_detections, distance)

    return remaining_detections
```

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.


0
25
50
75
100
125
150
175
200
0
50
100
150
200
250
300

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:

The transformation is a translation. The translation moves all points one unit to the right and one unit up. We can defined the transformation matrix as $M = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a rotation. The rotation rotates all points by 45°. We can defined the transformation matrix as $M = \begin{bmatrix} cos(45^\circ) & -sin(45^\circ) & 0 \\ sin(45^\circ) & cos(45^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a rotation. The rotation rotates all points by 90°. We can defined the transformation matrix as $M = \begin{bmatrix} cos(90^\circ) & -sin(90^\circ) & 0 \\ sin(90^\circ) & cos(90^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a scale. The scaling scales the x coordinates by 3 and the y coordinates by 5. We can defined the transformation matrix as $M = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a shear. The shearing increases the y coordinates by 5 times the x coordinate and the x coordinate is increased 3 times the y coordinate. We can defined the transformation matrix as $M = \begin{bmatrix} 1 & 3 & 0 \\ 5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a 90° rotation followed by a translation that moves all points to the left by one and up by one. Where the transformation can defined as $T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$ and the rotation can be defined as $R = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a translation followed by a reflection. The translation maps every point to a point that is three units to the left and one unit up. The reflection maps each point in the translated set to a point that is reflected across the line $y = x$. We can defined the transformation matrices as followed. $T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $R = \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a translation followed by scaling. The translation maps every point to a point that is three units to the right and five units up. The scaling factor here is 3 times the original. We can defined the transformation matrices as followed. $T = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix}$ and $S = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation is a translation followed by a rotation. The translation maps every point to a point that is six units to the left and 2 units up. The rotation is about the origin by 90° counter-clockwise. We can defined the transformation matrices as followed. $T = \begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$ and $R = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

The transformation involves shearing and then rotation. The shearing increases the y coordinates by 3 times the x coordinate and the x coordinate is increased 5 times the y coordinate. This is followed by a rotation by 90° counter-clockwise. We can defined the transformation matrices as followed. $S = \begin{bmatrix} 1 & 3 & 0 \\ 5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $R = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

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:

$$\mathrm{IoU}(\text{box}1, \text{box}2) = \frac{\text{Area of intersection of box1 and box2}}{\text{Area of union of box1 and box2}}.$$

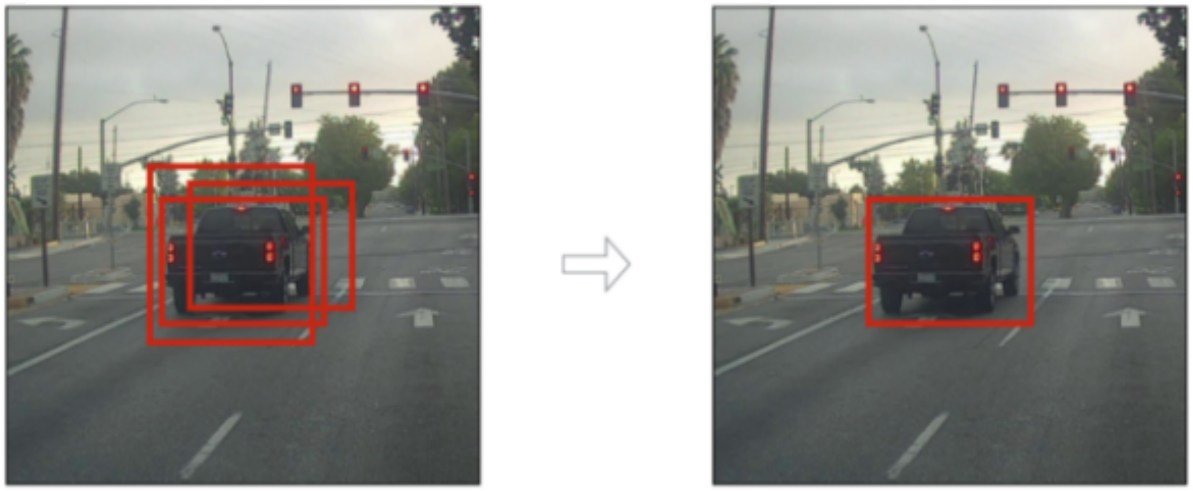


Figure 1: (Left) The object detection system identifies many candidates for the location of the car. For each candidate, there is a score which measures how likely it is to find a car in that bounding box. You can see that there are multiple red boxes, where each box will have a score between 0-1. (Right)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

The solution should employ the use of the Non-Maximal Suppression in order to suppress multiple detections. More specifically, we can begin by gauging what boxes count as overlaps by determining the intersection over union (IOU) between all boxes and setting a box as a local maximum. We can then sort these boxes to find the box with the highest score that contains most of all the other boxes.