DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, UCLA
ECE211A: DIGITAL IMAGE PROCESSING

INSTRUCTOR: Prof. Achuta Kadambi
TA: Pradyumna Chari

NAME: Peter Racioppo
UID: 103953689

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. All the problems need to be answered in the Overleaf document. Make a copy of the Overleaf project, and fill in your answers for the question in the solution boxes provided.

Submission

You will need to make two submissions comprising of PDF and code: (1) CCLE: You will save this Overleaf as PDF with all the answers (and necessary work) in boxes. We would like you to use LaTeX, but will accept handwritten or tablet notes. (2) You will also submit your Jupyter notebook (.ipynb file) with all the cells executed or a Matlab script with equivalent code. The course staff will provide only limited Matlab support.

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.

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

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.

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

1.3 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 $f^1(x)$.

$$\boxed{f(x+h) - f(x-h) = 2hf^{(1)}(x) + \tfrac{h^3}{3}f^{(3)}(x)}$$

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.

The unknowns are $f^{(1)}(x)$ 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 $f^{(1)}(x)$.

$f(x+h) - f(x-h) = 2hf^{(1)}(x) + \frac{h^3}{3}f^{(3)}(x)$
$f(x+2h) - f(x-2h) = 4hf^{(1)}(x) + 8\frac{h^3}{3}f^{(3)}(x)$

Multiplying the first equation by -2 and adding the two equations:
$-2f(x+h) + 2f(x-h) + f(x+2h) - f(x-2h) = -4hf^{(1)}(x) + 4hf^{(1)}(x) - 2\frac{h^3}{3}f^{(3)}(x) + 8\frac{h^3}{3}f^{(3)}(x)$

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

Plugging this back into the first equation:
$f^{(1)}(x) = \frac{1}{12h}(8f(x+h) - 8f(x-h) - f(x+2h) + f(x-2h))$

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.

$\frac{1}{12}\begin{bmatrix}1 & -8 & 0 & 8 & -1\end{bmatrix}$

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

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

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?

The unknowns are $f^{(2)}(x)$ 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 $f^{(2)}(x)$.

$f(x+h)+f(x-h) = 2f(x)+h^2 f^{(2)}(x)+\frac{h^4}{12}f^{(4)}(x)$
$f(x+2h)+f(x-2h) = 2f(x)+4h^2 f^{(2)}(x)+4\frac{h^4}{3}f^{(4)}(x)$

Multiplying the first equation by -4 and adding them,
$-4f(x+h)-4f(x-h)+f(x+2h)+f(x-2h) = -8f(x)-\frac{h^4}{3}f^{(4)}(x)+2f(x)+4\frac{h^4}{3}f^{(4)}(x)$

Solving for $f^{(4)}(x)$,
$f^{(4)}(x) = \frac{1}{h^4}(-4f(x+h)-4f(x-h)+f(x+2h)+f(x-2h)+6f(x))$

Plugging this back into the first equation,
$h^2 f^{(2)}(x) = f(x+h)+f(x-h)-2f(x)+\frac{1}{3}(x+h)+\frac{1}{3}f(x-h)-\frac{1}{12}f(x+2h)-\frac{1}{12}f(x-2h)-\frac{1}{2}f(x)$

$f^{(2)}(x) = \frac{1}{12h^2}(16f(x+h)+16f(x-h)-30f(x)-f(x+2h)-f(x-2h))$

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.

$$\boxed{\frac{1}{12}\begin{bmatrix}-1 & 16 & -30 & 16 & -1\end{bmatrix}}$$

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

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

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

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

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

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, $L(x,y)$.

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

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.

The normalization factor is $\int\int L(x,y)dxdy$. Letting $r^2 = x^2 + y^2$,
this comes to: $\frac{1}{2\pi\sigma^4}\int e^{-\frac{r^2}{2\sigma^2}}(\frac{r^2}{\sigma^2} - 2)rdrd\theta =$
$\frac{1}{\sigma^4}\int e^{-\frac{r^2}{2\sigma^2}}(\frac{r^2}{\sigma^2} - 2)rdr$
$= \frac{1}{2\sigma^4}\int e^{-\frac{r^2}{2\sigma^2}}(\frac{r^3}{\sigma^2})dr - \frac{2}{\sigma^4}\int e^{-\frac{r^2}{2\sigma^2}}(\frac{1}{\sigma^2})rdr = \frac{1}{2\sigma^4}\int e^{-\frac{r}{2\sigma^2}}(\frac{1}{\sigma^2})rdr - \frac{2}{\sigma^4}\int e^{-\frac{r^2}{2\sigma^2}}(\frac{1}{\sigma^2})rdr$
$= \frac{1}{\sigma^6}(-\sigma^2)e^{\frac{r}{2\sigma^2}}(r + 2\sigma^2) - \frac{2}{\sigma^4}\sigma^2(-e^{-\frac{r^2}{2\sigma^2}}) = \frac{1}{\sigma^2}$. So we should multiply $L(x,y)$ by σ^2.

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):
    mu = int((size-1)/2)
    LoG = np.zeros((size,size))
    for x in np.arange(size):
        for y in np.arange(size):
            LoG[x,y] = - (1/(np.pi*sigma**4)) \\
            *(1 - ((x-mu)**2 + (y-mu)**2)/(2*sigma**2)) \\
            *np.exp(-((x-mu)**2+(y-mu)**2)/(2*sigma**2))
    LoG /= np.sum(LoG)
    return LoG
```

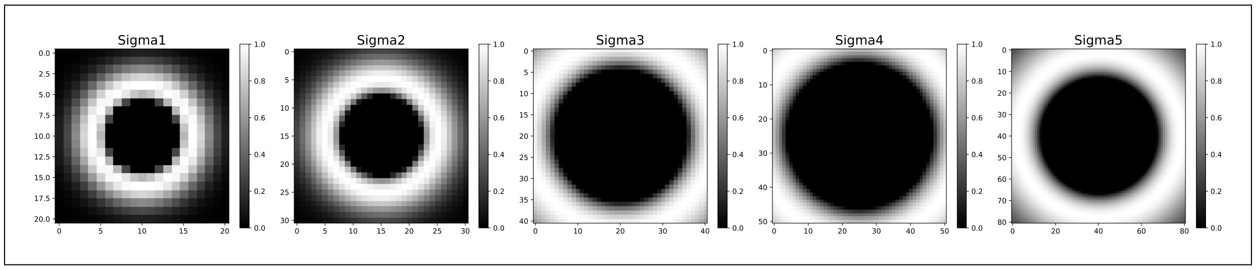
2.6 σ values (1.0 points)

(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.

$3, 7, 11, 15, 23$

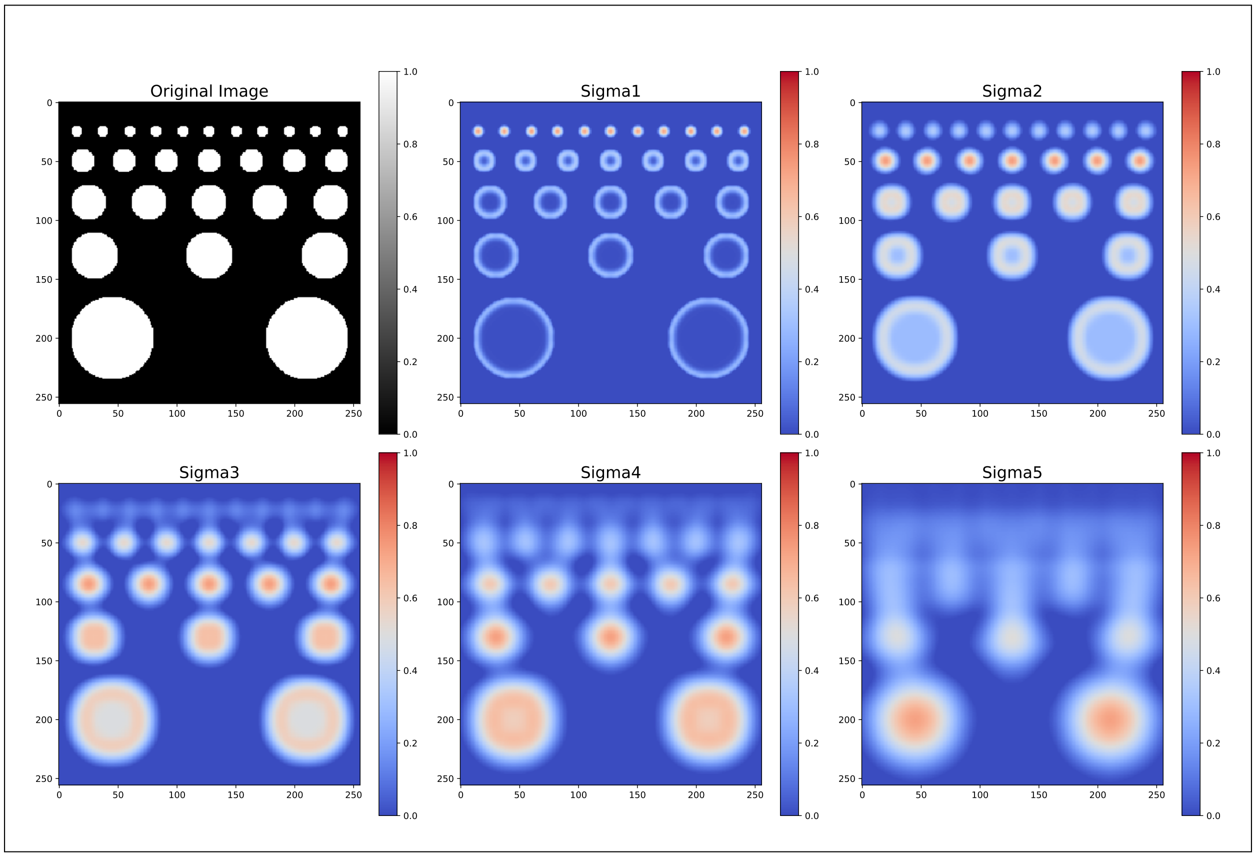
2.7 Visualize LoG Filter (1.0 points)

(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.



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.

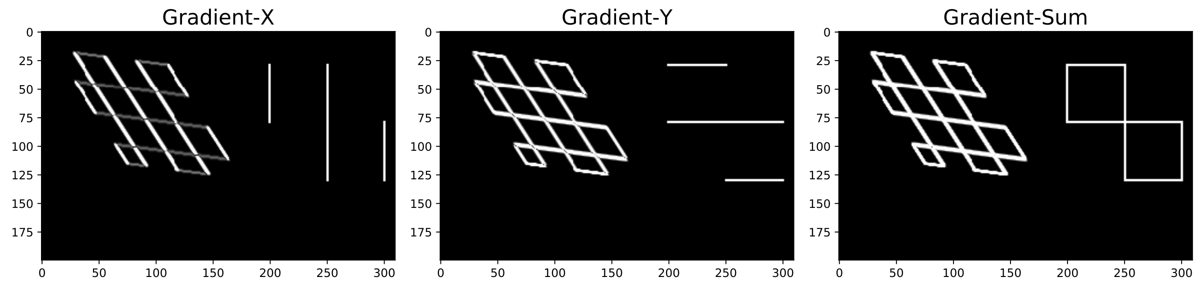
3.1 Computing Image Gradients Using Sobel Filter (1.0 points)

(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):
    dx = conv2D(img, sobel_x)
    dy = conv2D(img, sobel_y)
    return dx, dy
```

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):
    grad_x, grad_y = compute_image_gradient(image)
    Ixx = grad_x**2
    Ixy = grad_x*grad_y
    Iyy = grad_y**2
    return Ixx, Ixy, Iyy
```

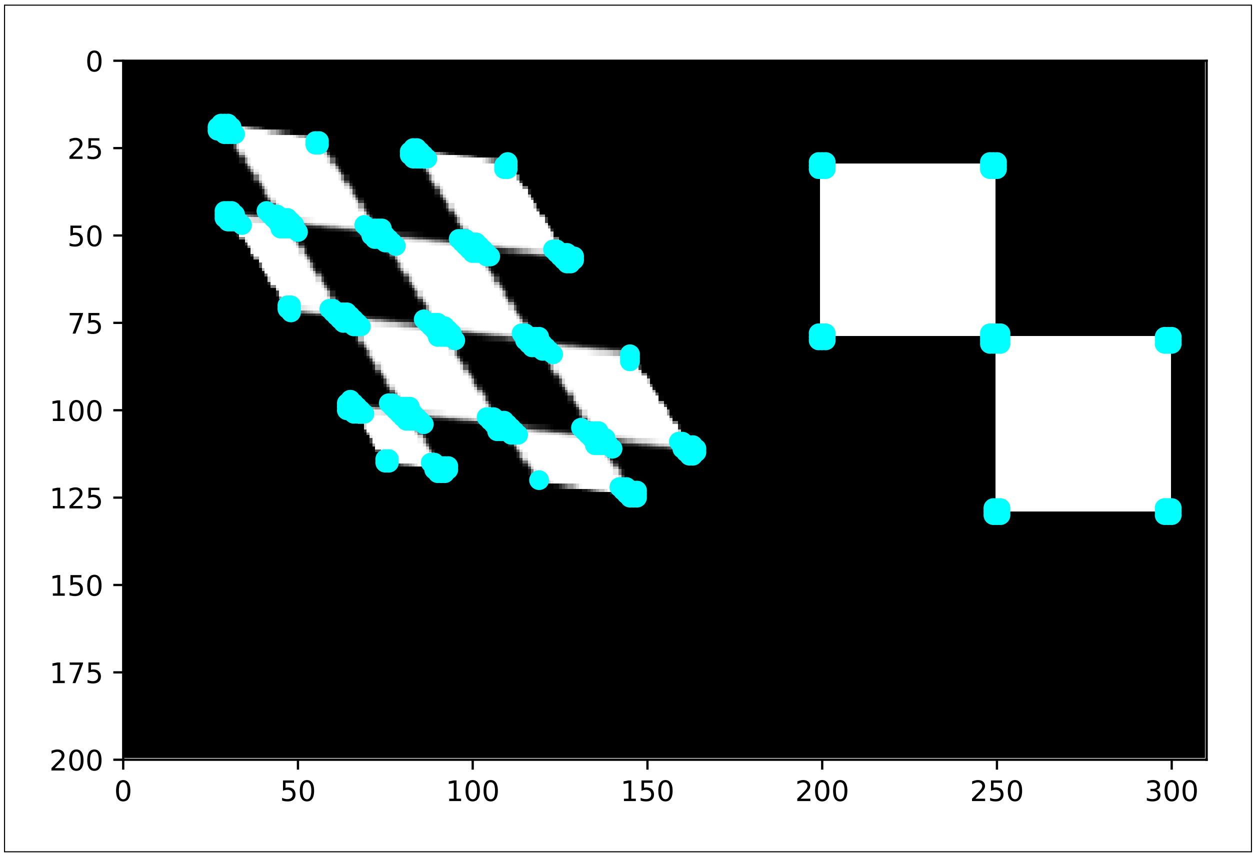
3.4 Harris Corner Response (1.0 points)

(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.

```
def harris_response(image: np.array, k: float, size: int):

    Ixx, Ixy, Iyy = grad_covariance(image, size)

    Sxx = conv2D(Ixx, average_filter(size))
    Syy = conv2D(Iyy, average_filter(size))
    Sxy = conv2D(Ixy, average_filter(size))

    R_img = Sxx*Syy-Sxy**2 - k*(Sxx + Syy)**2
    return R_img
```

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):
    x = candidate_detections[:,0]
    y = candidate_detections[:,1]
    h_v = H[x,y]
    h_argsort = np.flip(np.argsort(h_v))
    return candidate_detections[h_argsort]
```

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):
    N = np.shape(sorted_detections)[0]
    Max_x = []
    Max_y = []
    for i in np.arange(N):
        p1 = sorted_detections[i]
        Lx = np.array(())
        Ly = np.array(())
        for p2 in sorted_detections:
            if l2_distance(p1, p2) <= distance:
                Lx = np.append(Lx,p2[0])
                Ly = np.append(Ly,p2[1])

        am = np.argmax(H[Lx.astype(int),Ly.astype(int)])
        local_max = np.array([Lx[am],Ly[am]]).astype(int)
        Max_x.append(local_max[0])
        Max_y.append(local_max[1])
    m = np.shape(Max_x)[0]
    Max = np.zeros((m,2))
    Max[:,0] = Max_x
    Max[:,1] = Max_y
    return Max
```

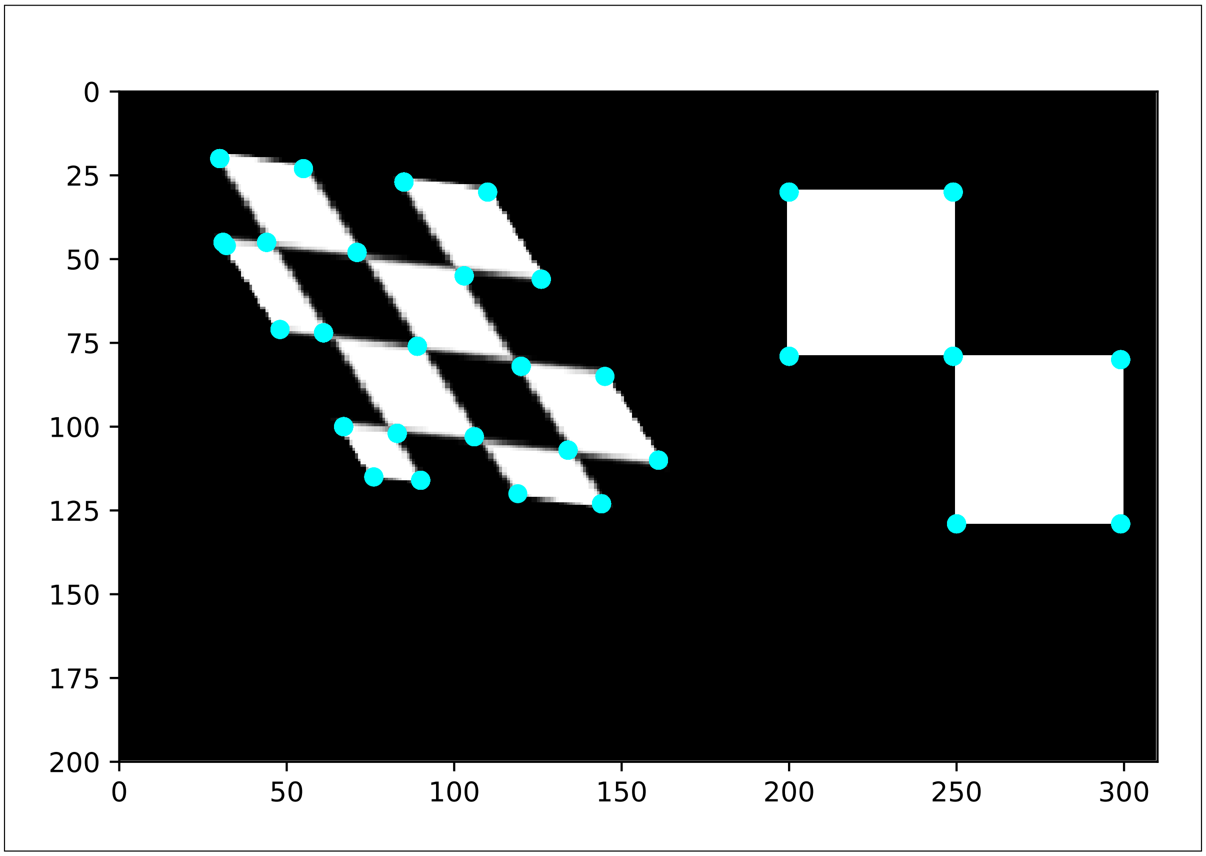
3.9 Non-Maximum Suppression: Putting it all together (1.0 points)

(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)
    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:

The transformation (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, which is a translation by 1 unit in the x direction and 1 unit in the y direction.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 0.71 & -0.71 & 0 \\ 0.71 & 0.71 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is a rotation by $\frac{\pi}{4}$.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is a rotation by $\frac{\pi}{2}$.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is a scaling by 3 in the x direction and 5 in the y direction.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 1 & 3 & 0 \\ 5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is a shear matrix with a shear of 3 in the x direction and 5 in the y direction.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, which is a rotation by $\frac{\pi}{2}$ followed by a translation by 1 unit in the x direction and 1 unit in the y direction.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, which is a rotation by $\frac{\pi}{2}$ followed by a translation by -1 unit in the x direction and 1 unit in the

y direction.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 3 & 0 & 1 \\ 0 & 5 & 1 \\ 0 & 0 & 1 \end{bmatrix}$, which is a scaling by 3 units in the x direction and 5 units in the y direction followed by a translation by 1 unit in the x direction and 1 unit in the y direction.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} 0 & -5 & 0 \\ 3 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is a scaling by 3 in the x direction and 5 in the y direction followed by a rotation by $\frac{\pi}{2}$.

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 (computed in Python; see the Jupyter notebook) is given by: $\begin{bmatrix} -5 & -1 & 0 \\ 1 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $= \begin{bmatrix} 1 & 3 & 0 \\ 5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, which is a rotation by $\frac{\pi}{2}$ followed by a shear by 3 in the x direction and 5 in the y direction.

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:

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

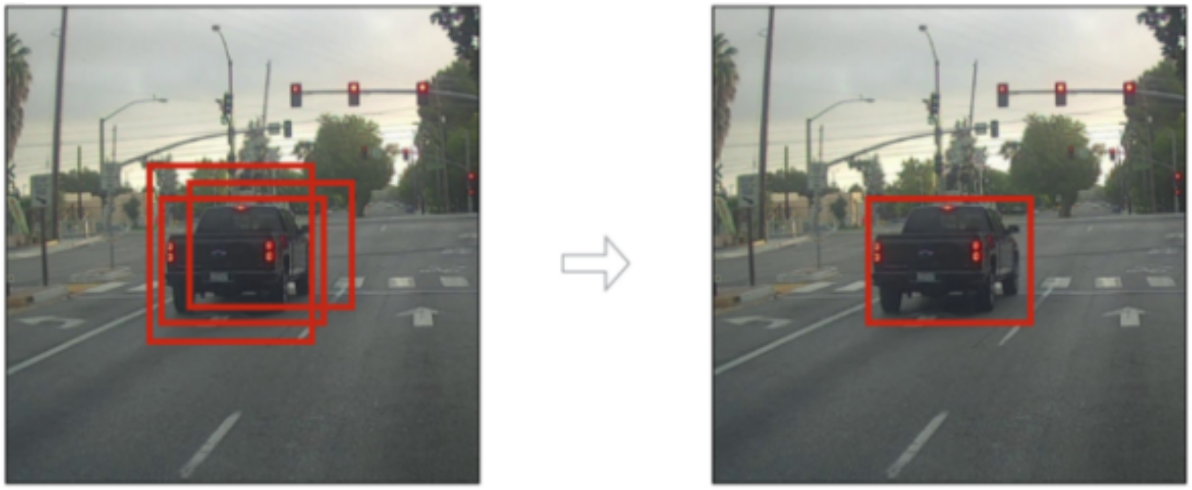


Figure 1: (Left) The object detection system identifies many candidates for the location of the car. For each candidate, there is a score which measures how likely it is to find a car in that bounding box. You can see that there are multiple red boxes, where each box will have a score between 0-1. (Right)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

We can use the non-maximum suppression technique. We simply sort the bounding boxes in order of highest scores and then remove the bounding boxes that have a high degree of overlap, as measured by the IoU, iterating until we have the desired number of boxes or have reached some minimum threshold on IoU values.