SCC.366 Media Coding and Processing
Course Work 2

This CW comprises **Three** distinct tasks:

- **Task 1:** Histogram of Oriented Gradient
- **Task 2:** Harris Corner Detector
- **Task 3:** Robustness of Harris Corner Detector

The submission deadline for submission is **Friday, 13 December 2024, 4:00 PM GMT**

Submission Requirements: For each of the first two tasks (Task 1 and Task 2), you are required to submit a single script that includes the function implementation, along with a few lines of code that demonstrate how to call the function, using default input parameters. For Task 3, you are not required to submit any code. Instead, you need to submit a report. The template for the report is provided in a separate file (i.e., "SCC.366-report-lastname-firstname.docx"). **The report should be no longer than one page**. Please ensure you follow the template closely, including the specified font size, font style, and heading formats.

This submission should consist of two MATLAB scripts, named "Task1.m" and "Task2.m," and a PDF report named "Report", all placed within a single folder named "your-first-name_your-last-name". Additionally, you need to include sample images that serve as default inputs for your functions.

Task 1: Histogram of Oriented Gradient

Introduction: A Histogram of Oriented Gradients (HOG) is a feature descriptor widely used in computer vision and image processing tasks, especially for object detection. It captures the distribution of intensity gradients and edge directions in an image.

Description: The HOG algorithm consists of the following steps:

- Gradient Computation: The first step in HOG involves computing the gradient of the image. An edge detector (use Sobel operator in this CW) is applied to the image to compute the gradient magnitude and orientation.

- Gradient Orientation Binning: The image is divided into small cells (use 8×8 in this CW), and for each pixel within a cell, the gradient magnitude and orientation are calculated. The gradient orientation is quantized into discrete bins (use 9 bins in this CW) to create an orientation histogram.

- Block Normalization: The cells are grouped into larger blocks (use 2×2 in this CW), and the histograms within each block are concatenated. This merging process should be done in an overlapping manner with a stride of one cell. To make the descriptor robust to changes in illumination and contrast, the block histograms are normalized. This can be done using different normalization techniques (use L2 in this CW).

- Sliding Window: The entire process is applied in a sliding window manner across the entire image to extract HOG features.

Task Requirements: You are required to write a MATLAB function, *'extractHOGFeatures'*, entirely from scratch. This function should be designed to calculate the Histogram of Oriented Gradients (HOG). This function should receive a grayscale image denoted as *inputImage* as input and return its corresponding Histogram of Oriented Gradients (HOG) features as the output. Ensure that both the height and width of the input image are divisible by 8; if not, resize the image accordingly.

MATLAB Function:

```
function extractedHOGFeatures = extractHOGFeatures (inputImage)
    % Your code here
end
```

Final MATLAB Script: The script should be formatted as follows:

```
% Load the input image
inputImage = imread('your_image.png');

% Ensure the image is grayscale

% Ensure image dimensions are divisible by 8, otherwise Resize the image

% Call the HOG feature extraction function
extractedHOGFeatures = extractHOGFeatures(inputImage);

% Display the length of the extracted features
disp(['Length of extracted HOG features: ',num2str(length(extractedHOGFeatures))]);
```

Input Validation: Ensure the input image is grayscale. If it is RGB, convert it using *rgb2gray*. Validate the dimensions of the image and handle resizing if necessary.

Note: You must implement your own code using basic MATLAB built-in functions, such as *imshow*, *subplot*, *linespace*, *zeros*, *imresize*, *tan, atan, atan2* and *im2double*. You are also allowed to use *imfilter* and *filter2* functions to apply a filter to the input image. Avoid using MATLAB built-in functions explicitly designed for generating kernels/filters, edges and histogram partitioning. Implement Sobel edge detector to extract edges. Additionally, minimize the use of loops to improve code efficiency.

Marks allocation for Task 1:

Code efficiency (e.g., avoid loops if possible, fewer lines) **10 marks**

Code commenting (descriptive comments for each line of the code as well as each code segment) **5 marks**

Code correctness and results **25 marks**

Task 2: Harris Corner Detector

Introduction: The Harris Corner Detector is a well-known image processing algorithm used to detect corners in an image. Corners are defined as points where there is a significant change in intensity in multiple directions, which often represent unique features in images. This task involves implementing the Harris Corner Detection algorithm using MATLAB. You will be required to compute the Harris response, identify potential corner candidates, and refine the results by checking for local maxima in a 3x3 window.

Description: The Harris Corner Detector involves several key steps:

1- **Derivatives**: The first step is to compute the image derivatives in both the x and y directions (I_x and I_y) using an edge detector. In this CW, use Sobel operator.
2- **Products**: These gradients are then used to compute three products ${I_x}^2$, $I_x I_y$ and ${I_y}^2$.
3- **Smoothing**: The products are smoothed using a Gaussian filter with kernel size of $w \times w$ and standard deviation (*sigma*) to generate A, B and C values for each pixel in the image.
4- **Harris Response**: The Harris response is then computed using the formula
$$R = A \times C - B \times B - k \times (A + C)^2$$
where k is a constant parameter between 0.04 and 0.06.
5- **Corner Candidates**: The (x,y)-th pixel becomes a potential corner candidate if its corner score is high enough:
$$R(x, y) > p \times \max(R),$$
where p is a constant parameter.
6- **Local Maxima**: Finally, the candidate pixel (x,y) becomes a corner if $R(x, y)$ is a local maximum in a 3×3 window.

Your task is to implement these steps using MATLAB, where various parameters such as w, $sigma, k$, p are provided as inputs.

Task Requirements: You are tasked with creating a MATLAB function *harrisCornerDetector*, which takes a grayscale image (*inputImage*) and several other parameters as inputs:

- w: Size of the Gaussian filter for smoothing.
- $sigma$: Standard deviation of the Gaussian filter for smoothing.
- k: The constant used in the Harris response calculation.
- p: The hyperparameter used to determine corner candidates.

The function should display the detected corners overlaid on the input image. An example output is shown below.

MATLAB Function:

```
function corners = harrisCornerDetector(inputImage, w, sigma, k, p)
    % Your function code here
end
```

Final MATLAB Script: The script should be formatted as follows:

```
% Load the input image
inputImage = imread('your_image.png');

% Ensure the image is grayscale

% Define parameters

% Call the Harris Corner Detector function
corners = harrisCornerDetector(inputImage, w, sigma, k, p);
```

Input Validation: The function should ensure that the input image is grayscale. If it is RGB, convert it to grayscale using *rgb2gray*. The function should validate that the *filterSize* is an odd integer.

Note: Avoid using MATLAB built-in functions for corner detection. Implement the Harris Corner Detector from scratch. You are expected to use basic MATLAB functions like *imshow*, *imread*, *subplot*, *max, imfilter* and *fspecial*. Minimize the use of loops to improve the code's efficiency.

Marks allocation for Task 2:

Code efficiency (e.g., avoid loops if possible, fewer lines) **10 marks**

Code commenting (descriptive comments for each line of the code as well as each code segment) **5 marks**

Code correctness and results **25 marks**

Task 3: Robustness of Harris Corner Detector

Introduction: In this task, you will evaluate the robustness of the Harris Corner Detector under four different scenarios: intensity shift, intensity scaling, image translation, and spatial scaling. The goal is to assess how the Harris Corner Detector performs when the image undergoes various transformations. You will manipulate the input image according to the given scenarios and apply the Harris Corner Detector to detect corners in each manipulated image. Finally, you will compare the results of the corner detection and evaluate the detector's robustness.

Task Requirements: Manipulate the Image. You will manipulate the input image using the following four scenarios:

- Intensity Shift**:** Shift the pixel intensities of the input image by adding or subtracting a constant value.

- Intensity Scaling**:** Scale the pixel intensities by multiplying them by a constant factor.

- Image Translation**:** Translate the image by shifting it in the x and/or y direction.

- Spatial Scaling**:** Resize the image by scaling it up or down.

For each manipulated image, apply your Harris Corner Detector function implemented in Task 2. Use the same parameters defined in Task 2 for all images. Visualize the detected corners overlaid on the original image for each manipulated version.

After detecting corners in each transformed image, evaluate the robustness of the Harris Corner Detector. Do the detected corners remain consistent across all transformations? How does each transformation affect the corner detection results? What happens to the number of corners detected after each transformation? For each scenario, write a few lines/paragraphs (follow the provided template) of the observed results. Discuss whether the Harris Corner Detector is theoretically and experimentally robust to each transformation and the potential reasons behind the observed behaviour.

Note: You are not required to submit any code for this task. Instead, you need to submit a report. The template for the report is provided in a separate file (i.e., "SCC.366-report-lastname-firstname.docx"). **The report should be no longer**

than one page. Please ensure you follow the template closely, including the specified font size, font style, and heading formats.

Marks allocation for Task 3:

Theory correctness (Correct understanding and explanation of the Harris Corner Detector's robustness in the context of the four image transformations)
5 marks

Experiment correctness (e.g., proper manipulation of the input image according to the four scenarios, correct application of the Harris Corner Detector)
10 marks

Correct Conclusion (A thoughtful, well-written conclusion for each scenario that discusses the impact of each transformation on the Harris Corner Detector's performance) **5 marks**