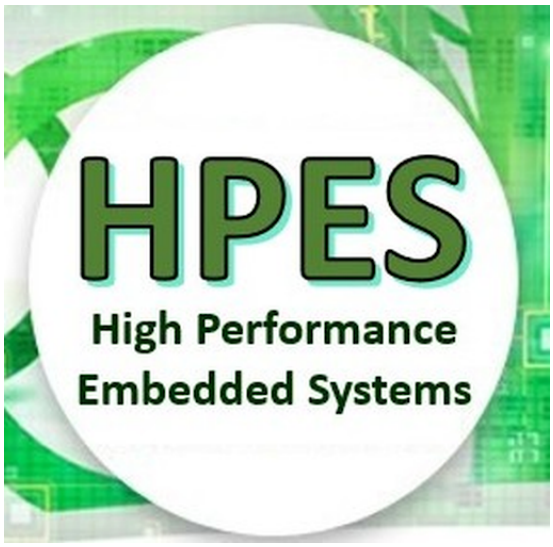

EEE4120F

High Performance Embedded Systems

Practical 1

Introduction to MATLAB/Octave & Performance Benchmarking

Course Team

Course Convener: Associate Professor Simon Winberg

Teaching Assistant: Travimadox Webb

University of Cape Town 2026

Practical 1

Introduction to MATLAB/Octave
& Performance Benchmarking

Summary of Deliverables

- **Duration:** 1 Week (20th Feb - 27th Feb)
- **Report:** A short technical report in IEEE Confrence Format (PDF, Max 2 Pages) via Gradescope assignment **Practical 1 Report**.
- **Code:** Submit your code(run_analysis.m) to Gradescope assignment **Practical 1 Code**.

Tip

The code for this practical can be found at: Practical-1

1 Objectives

The focus of this task is to familiarise you with **MATLAB** (or Octave) and the concepts of **benchmarking**. You are advised to use MATLAB since you are already familiar with it, though you are welcome to use the open source version i.e Octave.

In High Performance Embedded Systems, we often compare "Gold Standard" results (highly accurate, often slow) to high-speed approximations. This practical establishes the methodology for making those comparisons.

At the end of this of practical you will:

1. **Understand 2D Convolution:** Master the mathematical foundation and implementation
2. **Edge Detection:** Learn how filters detect edges in images
3. **Performance Analysis:** Compare manual vs. built-in implementations
4. **MATLAB Profiling:** Use timing functions to measure algorithm performance
5. **Image Processing Fundamentals:** Understand kernel operations and filtering

2 Introduction & Theory

Convolution is a fundamental operation in computer vision and signal processing. In this practical, you will explore using 2D convolution to perform edge detection in images. One of of the most common ways to perform edge detection is the Sobel Edge Detector.

The Sobel Edge detector makes use of a Sobel operator that performs 2-D spatial gradient measurement on an image emphasizing regions of high spatial frequency.Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

2.1 Sobel Operator

The Sobel operator is a fundamental tool in image processing used for edge detection. It consists of a pair of 3×3 convolution kernels, defined as follows:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{1}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2}$$

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid. One kernel handles each of the two perpendicular orientations.

2.1.1 Calculating the Gradient

The kernels are applied separately to the input image to produce separate measurements of the gradient component in each orientation (commonly denoted as G_x and G_y). These components can then be combined to find the absolute magnitude of the gradient at each point and the orientation of that gradient.

1. Gradient Magnitude

The exact magnitude of the gradient is calculated using the Pythagorean theorem:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

However, for computational efficiency, an **approximate magnitude** is often computed using absolute values. This method is much faster to compute while still providing sufficient accuracy for many applications:

$$|G| = |G_x| + |G_y|$$

2. Gradient Orientation

The angle of the edge's orientation (relative to the pixel grid) is determined by the arctangent of the ratio of the vertical to horizontal gradients:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

In this context, an orientation of 0 typically implies that the direction of maximum contrast runs from left to right, with other angles measured anti-clockwise from this reference.

2.2 Pixel-wise Computation

In practical implementations, the two components are often computed and added in a single pass over the image using the pixel neighborhood grid shown below.

Figure 2: Pixel Neighborhood Grid

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

Using this grid, where P_5 is the central pixel being processed, the approximate magnitude can be calculated directly by combining the weighted sums of the pixel intensities. The formula typically combines the vertical and horizontal difference calculations:

$$|G| = |(P_1 + 2P_2 + P_3) - (P_7 + 2P_8 + P_9)| + |(P_3 + 2P_6 + P_9) - (P_1 + 2P_4 + P_7)|$$

- **The first term** $|(P_1 + 2P_2 + P_3) - (P_7 + 2P_8 + P_9)|$ corresponds to the vertical gradient (applying the G_y kernel: Top row minus Bottom row).
- **The second term** $|(P_3 + 2P_6 + P_9) - (P_1 + 2P_4 + P_7)|$ corresponds to the horizontal gradient (applying the G_x kernel: Right column minus Left column).

3 Tasks

All code is to be implemented in the file: **run_analysis.m**

3.1 Task 1: Manual Convolution Implementation

Deliverable

Function to implement: `my_conv2`

Implement a custom function that accepts an input image and the Sobel kernels to perform 2D convolution.

1. You must implement the convolution logic manually using **only** `for` loops.
2. You are strictly prohibited from using any built-in convolution functions (e.g., `conv2`, `imfilter`) within this function.
3. Measure the execution time of your function using `tic` and `toc` for all images provided in the `sample_images` folder.
4. Handle image boundaries using appropriate padding.

Warning

The use of any inbuilt convolution functions (such as `conv2` or `imfilter`) to perform the core task will result in a **zero** being awarded.

While you must implement the algorithm manually, the padding you use is essential for verifying your results against MATLAB's built-in functions later. Ensure your manual logic matches the `conv2` documentation for:

- **'full'**: Returns the complete convolution.
- **'same'**: Returns the central part, matching the input image size.
- **'valid'**: Returns only the parts computed without zero-padded edges.

3.2 Task 2: Built-in Convolution Wrapper

Deliverable

Function to implement: `inbuilt_conv2`

Implement a wrapper function that takes an input image and the Sobel kernels, performing 2D convolution using the optimized built-in MATLAB/Octave function `conv2`.

1. Ensure the boundary handling (padding) options match your manual implementation for a fair comparison.
2. Measure the execution time using `tic` and `toc` for all images provided in the `sample_images` folder.

3.3 Task 3: Benchmarking Analysis

Deliverable

Function to implement: `run_analysis`

Perform a detailed benchmarking analysis comparing your manual implementation (`my_conv2`) against the built-in function (`inbuilt_conv2`).

- Load images from the `sample_images/` directory.
- Define Sobel operators (G_x and G_y) for horizontal and vertical edge detection.
- For each image:
  - Run both manual and built-in convolution implementations multiple times to ensure timing accuracy.
  - Measure execution time using the `tic` and `toc` functions.
  - Compute the performance gain: $Speedup = \frac{T_{manual}}{T_{builtin}}$.
  - Verify correctness by comparing the manual output against the built-in result.
  - Store results including image size, execution times, and calculated speedup.
- Analyse the performance difference.
- Comment on the scalability of each approach as image size increases.

Report Requirements

Format your report as a technical article. Include the following sections:

- **Introduction:** Briefly outline the practical aim, your implementation strategy, and the benchmarking goals.
- **Methodology:** Describe the your manual implementation approach, and the timing methodology.
- **Results:** Present tables or graphs comparing performance across different image sizes.
- **Discussion:** Analyze the performance difference.
- **Conclusion:** Summarise findings, identify limitations, and suggest potential improvements.

4 Marking Rubric

Category	Component	Marks
Implementation (40%)	`my_conv2` (Manual)	20
	`inbuilt_conv2` (Wrapper)	10
	Code Quality & Verification	10
Report (60%)	Introduction	5
	Methodology	10
	Results Presentation	10
	Discussion	**25**
	Conclusion	5
	Formatting	5
Total		**100**

Table 1: Mark Allocation for Practical 1