Introduction to Computer Vision

Final Projects

ESIN, UIR — Spring 2026

Ilias TOUGUI

Introduction

Throughout this course, you have acquired a solid foundation in computer vision, ranging from fundamental concepts such as linear filtering and convolution to advanced topics like Fourier transform and feature detection (edges, corners, and boundaries). The following projects are designed to allow you to apply these concepts in a practical, hands-on manner.

Depending on the project you choose, you may also encounter new concepts that were not covered in class (e.g., perspective transforms, homography). Learning these independently is part of the project experience—it reflects real-world practice, where engineers often need to explore and integrate new ideas. Each project description clearly separates the concepts from the course and the new ones you will explore.

Implementation Freedom

You may implement the projects entirely from scratch (using only Python and NumPy) or leverage OpenCV for certain tasks. The choice is yours, but the core algorithms that are central to the project should be understood and, where appropriate, implemented by you. Using OpenCV's high-level functions is acceptable if you explain the underlying principles in your report.

Deliverables

Your project submission must be a GitHub repository containing the following:

- **data/** – A folder with the input images you used. Include a short `README.txt` describing the data.
- **code/** – One Jupyter notebook (`.ipynb`) containing your code. The code must be:
  - Clean, well-commented, and organised into functions and classes.
  - Each function should have clear inputs and outputs (docstrings recommended).
  - When executed sequentially, the notebook should run a complete pipeline from loading data to producing final output.
  - If your project requires an interactive interface (e.g., for applying filters), you may use `ipywidgets` within the notebook to create a simple GUI, or provide an additional Python script alongside the notebook. The notebook itself must still demonstrate the full pipeline with at least one example.
- **report.pdf** – A short report (2–4 pages) explaining your approach, the challenges you faced, and how you evaluated your results. Include relevant figures and references to course concepts or new concepts that you have learned.

General Guidelines

- All projects must be submitted via GitHub. The link to the repository should be shared publicly via the Submission Form.
- The Jupyter notebook should be self-contained: when run from start to finish, it should load the data, execute the pipeline, and display/save the final results.
- The report (PDF) must be included in the root of the repository.
- Projects should be done in groups (a maximum of 4) from the same Lab Group.
- Late submissions will be penalized.
- Absence on the day of the defense is not permitted.

Project Difficulty & Maximum Marks

Projects vary in complexity. The maximum grade attainable depends on the project you choose:

Project	Topic	Difficulty	Max mark
Project 1	Document Scanner	• Accessible	16/20
Project 2	Instagram-Style Filters	• Accessible	16/20
Project 3	Barcode & QR Code Scanner	• Intermediate	18/20
Project 4	Panorama Stitcher	• Intermediate	18/20
Project 5	License Plate Deblurring	• Challenging	20/20

A perfect submission of a harder project will always outscore a perfect submission of an easier one. We encourage you to reach.

Project 1: Document Scanner

Accessible Max: 16/20

Summary

Develop a pipeline that automatically detects a document (e.g., a book page, receipt, or whiteboard photo) from an image, corrects its perspective to obtain a flat top-down view, and enhances the result for improved readability. The result should be an interactive GUI application that allows users to load an image, scan the document, and save the output. This is a classic computer vision application widely used in scanning and office automation.

Tasks

- Find or capture a photo of a document to use as input.
- Preprocess the image (grayscale, Gaussian blur).
- Apply edge detection (Canny) to extract edges.
- Find the largest quadrilateral contour that represents the document using contour analysis.
- Order the four corner points consistently.
- Apply a perspective transform to obtain a top-down view.
- Enhance the result (e.g., histogram equalisation, gamma correction).

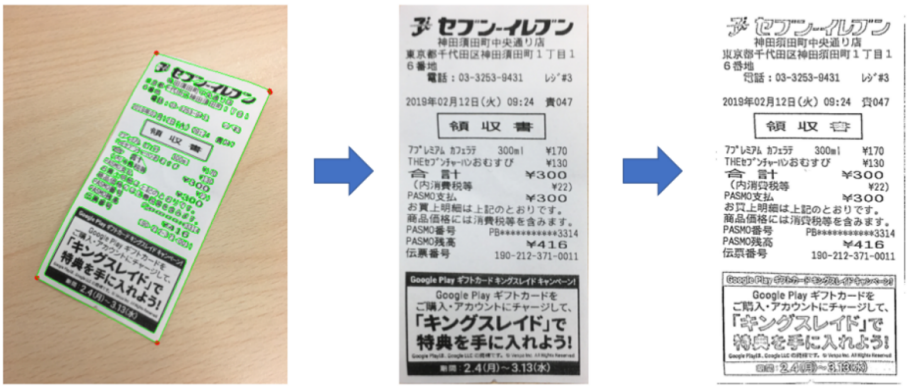

Figure 1: Top-down scanned view

Project 2: Instagram-Style Filters

Accessible Max: 16/20

Summary

Create a set of photo filters inspired by social media apps. You will implement both linear and non-linear filters, colour transformations, and blending effects. The result should be an interactive application (e.g., a Jupyter notebook with `ipywidgets` or a simple GUI) that allows users to apply various filters to their images.

Tasks

- Implement 2D convolution to apply custom kernels (e.g., blur, sharpen, emboss).
- Implement non-linear filters: median filter and bilateral filter (edge-preserving smoothing).
- Apply colour transformations: sepia tone or colour space manipulations.
- Create special effects: vignette (blending with a radial gradient), pencil sketch (edge detection + inversion), or oil painting effect.
- Combine the filters into a simple interface (e.g., using `ipywidgets` or a command-line script) to load an image, select a filter, and save the result.

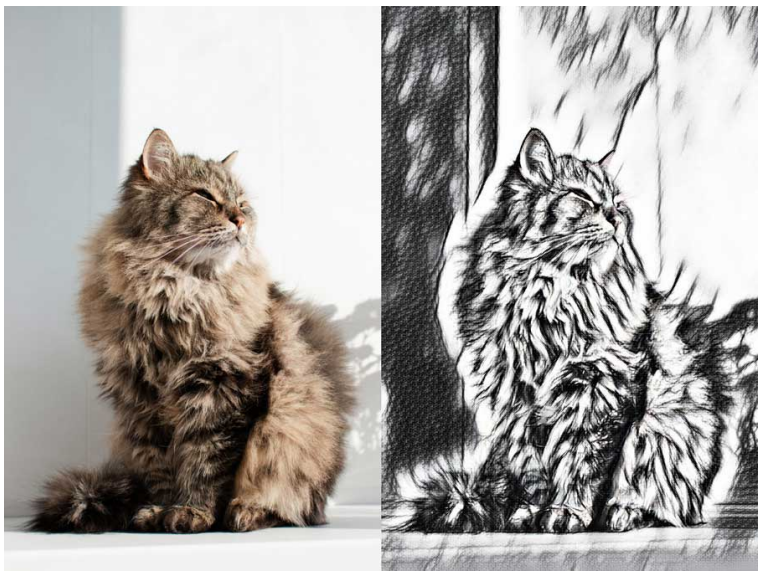

Figure 2: Pencil sketch filter example

Project 3: Barcode and QR Code Scanner and Localizer

Intermediate Max: 18/20

Summary

Create a system that detects, localises, and decodes barcodes and QR codes from images. You may implement the detection pipeline from scratch using fundamental image processing techniques, or leverage OpenCV's built-in detectors. Regardless of the approach, you must demonstrate a clear understanding of the underlying principles in your report. Implement the system as an interactive GUI application that allows users to load an image, scan for codes, and visualise the results.

Tasks

- Find or capture images containing barcodes and QR codes to use as input.
- Detect and localise each code (draw bounding boxes or quadrilaterals).
- Decode the information contained in the code.
- If implementing from scratch, use techniques such as edge detection, contour analysis, and geometric reasoning.
- If using OpenCV's built-in functions, explain the algorithms they replace in your report.
- Visualise the results and include example outputs in your submission.
- Combine the pipeline into a simple GUI application to load an image, scan for codes, and display the decoded results.

Figure 3: Barcode/QR code Scanner example

Project 4: Panorama Stitcher

Intermediate Max: 18/20

Summary

Stitch two or more overlapping images into a seamless panorama. This project combines feature detection, matching, homography estimation, and blending. You may implement key steps from scratch or use OpenCV for the geometric transformations. Find or capture your own overlapping images to use as input, or create synthetic ones.

Tasks

- Find or capture sets of overlapping images to stitch (e.g., a panoramic scene), or create synthetic overlapping images.
- Detect keypoints using the Harris corner detector or SIFT (you may implement Harris from scratch or use OpenCV's version).
- For each keypoint, extract a local patch and create a simple descriptor (e.g., raw pixel intensities, a gradient histogram, or SIFT descriptors).
- Match keypoints between images using normalised cross-correlation, SSD, or nearest-neighbour matching.
- Estimate a homography using `cv2.findHomography` with RANSAC.
- Warp the images to a common coordinate system.
- Blend the overlapping regions (simple alpha blending or, optionally, multi-band blending in the frequency domain).

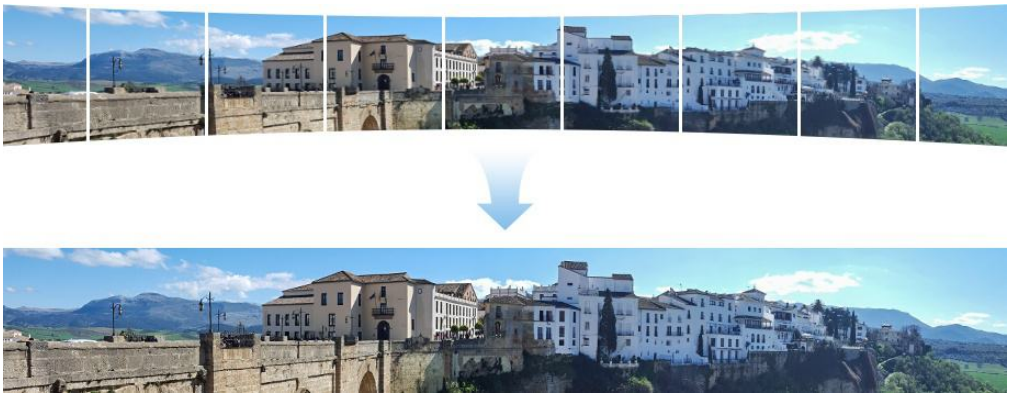

Figure 4: Panorama Stitcher example

Project 5: License Plate Deblurring

Challenging Max: 20/20

Summary

Develop a pipeline that restores sharp, readable license plates from motion-blurred images. Motion blur frequently occurs when vehicles are moving at high speeds, making license plate recognition challenging. This project applies deconvolution techniques to reverse the effects of motion blur, directly building on your course material on deconvolution and Wiener filtering. You will estimate the blur kernel (Point Spread Function) and then apply frequency-domain deconvolution to recover a clear image.

Tasks

- Generate synthetic motion-blurred images with known blur parameters (length and angle) to create a training/validation dataset.
- Estimate the motion blur parameters using frequency domain analysis (Fourier transform magnitude patterns) or OPTIONALLY via a simple CNN model that takes the FFT of the blurred image as input.
- Implement the Wiener filter in the frequency domain to deconvolve the image using the estimated PSF.
- Evaluate deblurring quality using metrics such as PSNR (Peak Signal-to-Noise Ratio).
- Optionally, extend to real-world blurred license plate images and compare results with standard image deblurring techniques.

Figure 5: License Plate Deblurring example

Good luck and have fun building your own computer vision applications!