IITB Mars Rover Team

Freshie Induction 2025

Software Subsystem

Assignment 4 - Classical Computer Vision

Deadline: 5th January 2026

This assignment is designed to build strong intuition in **Classical Computer Vision**. You are expected to design pipelines, justify choices, and analyze outcomes, not just apply individual algorithms.

How to Approach This Assignment?

The primary aim of this assignment is to develop a strong intuition for Classical Computer Vision. You are encouraged to experiment broadly rather than settling early on a single algorithm that appears to work.

- Do not restrict yourself to one method per task. Experiment with multiple algorithms and compare their behavior, strengths, and failure cases.
- Refer to the session slides for the core concepts expected to be used. You may also browse the internet for additional resources, variations, and explanations.
- You may use AI tools primarily for code generation and syntax assistance. However, the pipeline design, logic, parameter choices, and explanations must be your own. You should be able to reasonably justify every major decision in your pipeline.
- Perform documentation alongside experimentation. Maintain notes of observations, intermediate results, parameter changes, and unexpected behaviors. This will greatly help in analysis and reporting.

Recommended Environment

For experimenting with computer vision pipelines, it is strongly recommended to use an interactive notebook-based environment such as:

- Google Colab (cloud-based)
- Jupyter Notebook (local setup)

These platforms allow:

- Step-by-step execution without rerunning the entire pipeline
- Easy visualization of intermediate outputs
- Quick iteration over parameters and algorithms
- Saving images, plots, and results alongside code

Getting started:

- To get started with Jupyter Notebook, watch: Introduction to Jupyter Notebook
- Refer to this starter tutorial notebook: Classical CV Tutorial Notebook

It is highly recommended to visualize intermediate outputs at every major processing step to understand how each operation transforms the image.

Submission Guidelines

- Submit a single PDF report
- Include visual results, explanations, and observations
- Code snippets are optional but encouraged
- Evaluation prioritizes clarity of reasoning over final performance

ALL THE BEST!!

Exercise 1: Spot the Mallet!

Several videos containing a **mallet** captured using a **moving camera** are provided in the directory:

`/Mallet_videos`

Your task is to segment the mallet from each video frame. The key requirement is to design a single robust pipeline that works across all the provided videos!

Note: This exercise is not about using one perfect threshold value for one video, but about understanding how preprocessing, color spaces, and thresholding interact to produce stable segmentation results.

Objective

Design an end-to-end video processing pipeline that:

- Reads and processes video frames
- Isolates the mallet as a binary mask
- Cleans and refines the segmentation
- Visualizes intermediate and final results

Implementation

- **Video Handling**
  - Load and play the videos using OpenCV
  - Extract and process frames sequentially
  - Observe and comment on variations across videos (lighting, background, motion)
- **Color Space Exploration**
  - Convert frames from RGB/BGR to alternative color spaces.
  - Analyze which channels best separate the mallet from the background
  - Justify your chosen color space for segmentation
- **Preprocessing:** Apply and compare preprocessing techniques such as:
  - Gamma correction
  - Contrast enhancement
  - Intensity normalization

  Visualize the effect of each step and explain why certain preprocessing steps improve thresholding performance.
- **Thresholding-based Segmentation**

  - Use thresholding as the primary segmentation tool
  - Experiment with:
    * Global thresholding
    * Adaptive thresholding
  - Compare their robustness across videos
- **Binary Mask Refinement**
  - Clean the binary mask using morphological operations
  - Explain the choice of:
    * Operation (erosion, dilation, opening, closing)
    * Structuring element shape and size
- **Final Segmentation and Visualization**
  - Extract the final mallet mask
  - Apply the mask to the original frame
  - Visualize:
    * Original frame
    * Binary mask
    * Masked output

You are encouraged to modify, reorder, or extend this pipeline based on your observations.

Bonus: Segmentation using K-Means Clustering (Optional)

As an alternative to thresholding-based segmentation, explore the use of **K-Means clustering** for segmenting the mallet.

- Apply K-Means clustering on pixel features such as:
  - Color values (in RGB, HSV, or LAB space)
  - Intensity or selected channel values
  - (Optional) Spatial coordinates
- Experiment with different values of K and feature combinations.
- Identify the cluster corresponding to the mallet and generate a binary mask.
- Compare the K-Means based segmentation with your thresholding-based approach in terms of:
  - Robustness across videos

  - Sensitivity to illumination changes
  - Ease of parameter tuning
  - Computational cost

Discussion and Analysis

In your report, clearly discuss:

- A clear high-level overview of your complete segmentation pipeline
- Why the proposed pipeline works consistently across multiple videos
- Failure cases and limitations of the approach
- Sensitivity of the pipeline to lighting variations and background changes

Exercise 2: Cone Detection in the Wild

Multiple images containing **cones** in **diverse environments** are provided in the directory:

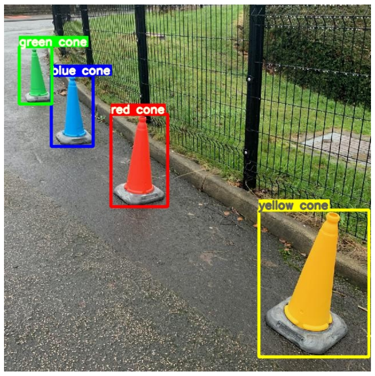
`/Cone_images`

Your task is to detect and localize all cones present in each image frame. The key challenge is to design a unified robust pipeline that generalizes across varying backgrounds, lighting conditions, and cone placements. There are images with multiple cones too.
Note: This exercise emphasizes building a complete detection system, from low-level pixel processing to high-level object localization, using classical computer vision techniques.

Objective

Design an end-to-end cone detection pipeline that:

- Processes images to isolate regions with the cone.
- Extracts spatial structure using edges and contours.
- Filters and validates detections based on geometric properties.
- Produces bounding boxes or contours around detected cones.
- Handles multiple cones per image and varying scene complexity.



Implementation

- **Image Handling and Initial Observations**
  - Load and visualize the cone images using OpenCV
  - Process the images sequentially and observe challenges:
    - Multiple cones at different distances
    - Varying backgrounds (roads, grass, indoor/outdoor)
    - Lighting variations (shadows, highlights, overexposure)
    - Partial occlusions and perspective distortions
  - Document these observations, they will guide your pipeline design
- **Color Space Analysis and Selection**
  - Convert images to multiple color spaces (HSV, LAB, YCrCb, etc.)
  - Analyze channel-wise separability of cone color from background
  - Create visualizations showing:
    - Individual channel histograms
    - Channel images for different color spaces

    - Color distribution of cone pixels vs. background pixels
  - Select and justify the most discriminative color space(s) for cone segmentation
  - Consider using multiple color spaces in combination
- **Preprocessing Pipeline**
  - Apply preprocessing to enhance cone visibility:
    - Noise reduction (Gaussian blur, bilateral filtering)
    - Illumination normalization techniques
    - Contrast enhancement (CLAHE, histogram equalization)
  - Visualize the impact of each preprocessing step
  - Explain which preprocessing operations improve subsequent detection stages
- **Edge-Based Isolation of Cone Boundaries**
  - Apply edge detection algorithms to isolate cone boundaries:
    - Canny edge detector
    - Sobel/Scharr gradient operators
    - LoG/DoG
    - (Optional) Edge extraction using Histogram of Oriented Gradients (HOG) to enhance structural boundary representation
  - Compare edge responses between:
    - Original frames
    - Preprocessed frames (noise reduced / contrast enhanced)
  - Tune edge detector parameters (thresholds, kernel sizes), and justify choices based on maintaining cone boundaries while suppressing background edges
  - Discuss why edge-only output is not yet sufficient for cone segmentation
- **Edge Enhancement and Structural Line Extraction**
  - Improve the quality and continuity of detected edges using morphological operations:
    - Dilation to strengthen weak or broken cone boundaries
    - Closing to bridge small gaps in edge segments
    - Removal of isolated noisy edge fragments
  - Analyze how morphological operations affect:
    - Edge continuity
    - Boundary sharpness

    * Suppression of background clutter
  - Apply the Hough Line Transform on the refined edge map to detect prominent straight-line structures:
    * Identify candidate cone edges based on line orientation and length
    * Exploit the characteristic slanted edges of traffic cones
    * Filter detected lines using angular and spatial constraints
  - Visualize and compare:
    * Raw edge output
    * Morphologically enhanced edges
    * Detected Hough lines overlaid on the image
  - Discuss the role of line-based detection in narrowing down cone candidates and why structural reasoning is beneficial before region-based analysis

- **Contour Detection and Geometric Verification**
  - Extract contours from the refined cone candidate regions
  - Filter candidate cone contours using geometric constraints:
    * Area (minimum/maximum threshold)
    * Perimeter and compactness
    * Aspect ratio (height-to-width ratio of cone silhouette)
    * Solidity (contour area vs. convex hull area)
    * Circularity or other shape descriptors
  - Justify the geometric constraints based on physical cone characteristics
  - Handle frames with multiple detected regions (multiple cones)
- **Cone Localization, Color Identification, and Visualization**
  - For each validated contour:
    * Compute bounding boxes or fit contours
    * Extract centroid positions
    * Determine the dominant color of the cone region (e.g., orange, red, blue, yellow)
    * Optionally estimate cone orientation
  - Visualize detection results:
    * Original frame
    * Edge-detected image or a custom edge-detection kernel

    * Refined binary mask with contours
    * Final output with bounding boxes/contours overlaid
  - Display multiple visualization stages side-by-side for comparison
- **Pipeline Integration and Testing**
  - Integrate all stages into a cohesive processing pipeline
  - Test the complete pipeline on the provided images
  - Quantitatively or qualitatively evaluate:
    * Detection rate (true positives)
    * False positives (non-cone detections)
    * Robustness to background changes
  - Identify and document failure cases with explanations

You are encouraged to iterate, refine, and extend this pipeline. Consider hybrid approaches that combine multiple techniques for improved robustness.

Discussion and Analysis

In your report, provide a comprehensive analysis including:

- **Pipeline Architecture:**
  - High-level flowchart of your detection pipeline
  - Explanation of how each stage contributes to final detection
  - Discussion of the interplay between color, edge, and shape information
- Color Space Justification
- Edge Detection Strategy
- Contour Filtering Criteria
- Robustness and Generalization
- Failure Analysis

Exercise 3: Clean and Detect the ArUco Marker!

A set of videos containing **ArUco markers** with different levels of noise, blur, and lighting variation are provided inside:

`/ArUco_videos`

Note: Each video file follows the naming convention: `Video{S.No.}({Dictionary}).mp4`.

- The dictionary name embedded in the filename indicates the ArUco dictionary (e.g., `DICT_4X4_50`, `DICT_5X5_100`) that should be used for detection.
- You are expected to parse this information and initialize the corresponding OpenCV ArUco dictionary for each video.

Your task is to explore classical image filtering using ernels and convolution, progressively design a denoising pipeline, and finally detect the ArUco marker consistently across videos.

Before You Begin: ArUco Detection in OpenCV

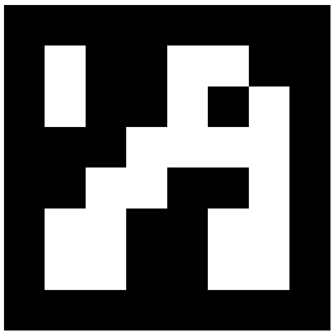
To understand how ArUco markers are normally detected, review this short OpenCV guide.

- Try running a basic example detection script before applying any denoising.
- Observe: In which frames does the marker fail due to noise/blur?

Objective

Develop an end-to-end video processing system that:

- Reads and processes noisy video frames
- Removes noise using convolution-based filtering
- Enhances visibility of the marker
- Detects the marker using OpenCV's ArUco module
- Visualizes intermediate filter results and final detection output



Implementation

- **Video Handling**
  - Load the video using OpenCV
  - Process frames sequentially
  - Identify types of degradation: noise, motion blur, defocus blur, low contrast
  - Note how these affect marker edges and corner sharpness
- **Understanding Kernels and Convolution**
  - Manually define custom convolution kernels as NumPy arrays

    - Apply them using 2D convolution (`cv::filter2D`)
    - Experiment with:
      - Kernel size and symmetry
      - Padding strategies (zero, reflect, replicate)
      - Effect of kernel normalization
    - Observe how convolution modifies local image structure
  - **Noise Reduction vs Edge Preservation**
    - Experiment with smoothing filters:
      - Mean (box) filtering
      - Gaussian blurring
      - Median filtering
      - Bilateral filtering
    - Analyze trade-offs between noise removal and edge degradation
    - Identify situations where smoothing harms ArUco detectability
  - **Sharpening and Edge-Enhancing Filters (Core Focus)**
    - Design and apply custom sharpening kernels:
      - Laplacian-based sharpening
      - Unsharp masking
      - High-boost filtering
    - Compare built-in vs custom-designed kernels
    - Visualize how sharpening recovers square boundaries and corners
    - Justify kernel choices based on marker geometry
  - **Edge Detection for Structural Enhancement (Optional but Recommended)**
    - Apply edge detectors:
      - Sobel / Scharr gradients
      - Canny edge detector
      - Laplacian of Gaussian (LoG)
    - Analyze how edge maps correlate with ArUco borders
    - Discuss why edge detection alone is insufficient, but useful for validation
  - **Contrast Normalization and Thresholding**

    * Convert frames to grayscale
    * Apply contrast enhancement (CLAHE / histogram equalization)
    * Compare adaptive and global thresholding
    * Evaluate how thresholding quality affects marker decoding

- **Final ArUco Detection**
    * Initialize the correct ArUco dictionary from the video name
    * Tune detector parameters (adaptive threshold window, corner refinement)
    * Detect marker corners and IDs
    * Overlay bounding box and ID on original frames
- **Visualization Output** Display and compare:
    * Original frame
    * Smoothed frame
    * Sharpened / edge-enhanced frame
    * Edge map (if used)
    * Final ArUco detection result

Discussion and Analysis

In your report, address:

- Which convolution kernel(s) performed best and why
- How noise level influenced ArUco detection
- Comparison of filtering methods: edge-preserving vs smoothing
- Failure cases (e.g., extreme blur, very low contrast)
- How detection confidence might change after filtering

***