# OpenCV Assignment

Mihika Maheshwari

January 2026

---

## 1 Exercise 1

### 1.1 Overview of Pipeline

The video is processed over each frame which is converted to LAB colour space. While both LAB and HSV distinctively highlight the mallet, HSV does so more prominently. However, individual HSV channels, like saturation, are more influenced by shadows, whereas the B channel from LAB highlights the mallet without capturing shadows.

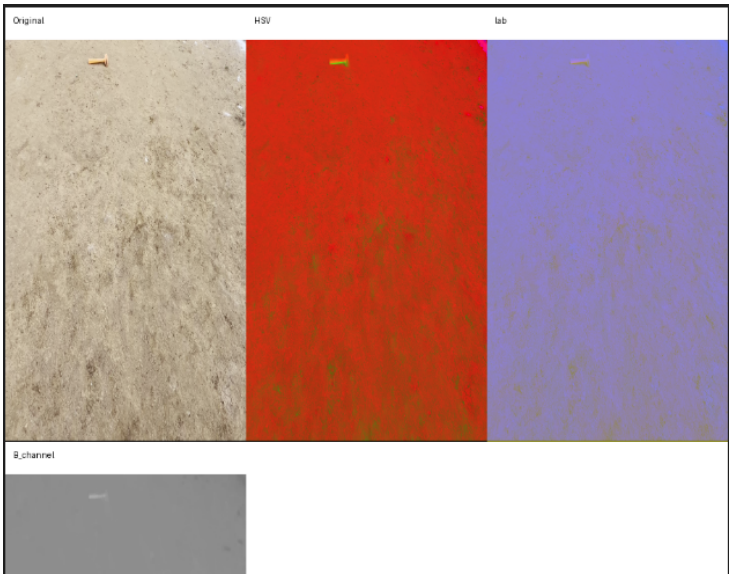



**Figure 1:** Channel

**Pre-processing steps:**

1. Gaussian Blur: to blur the sand texture
2. Gamma Correction
3. Intensity Normalisation: to highlight the bright orange colour

Contrast enhancement was not used as it amplified the sand texture rather than improving mallet detection.

Both binary thresholding and Otsu thresholding were applied to mask out the mallet. Binary thresholding was sometimes less effective than Otsu. However, under sunlight or when some texture remained after pre-processing, Otsu would fail, masking almost the entire frame. To mitigate this, a combined mask was created using a bitwise AND operator.

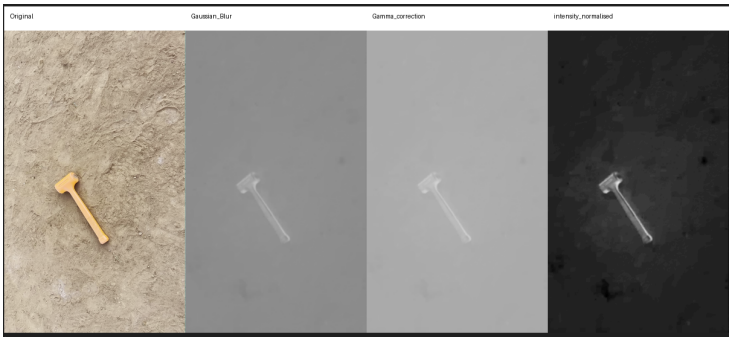
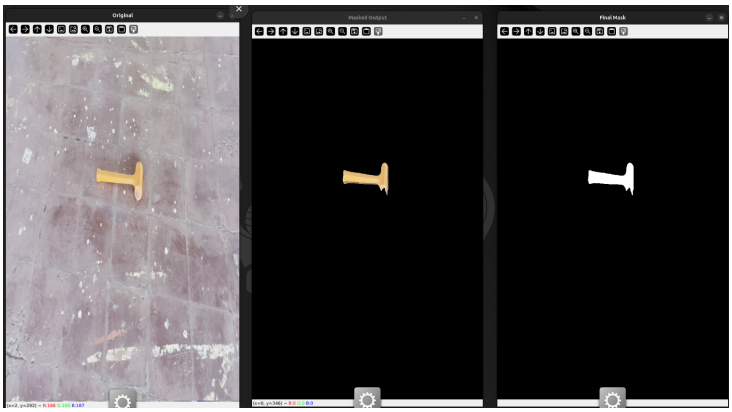



**Figure 2:** Pre Processing

Adaptive thresholding was not used because it separated the shadow of the mallet by detecting local intensity differences.

**Figure 3:** Adaptive Thresholding - shows shadow separation issues

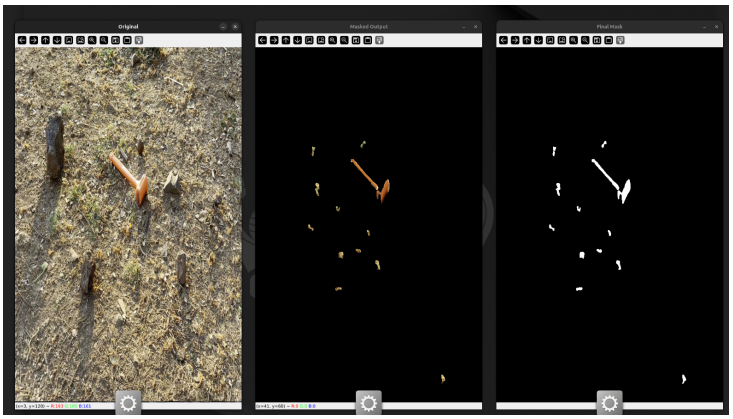
Finally, morphological opening and closing refined the mask to produce the final result.



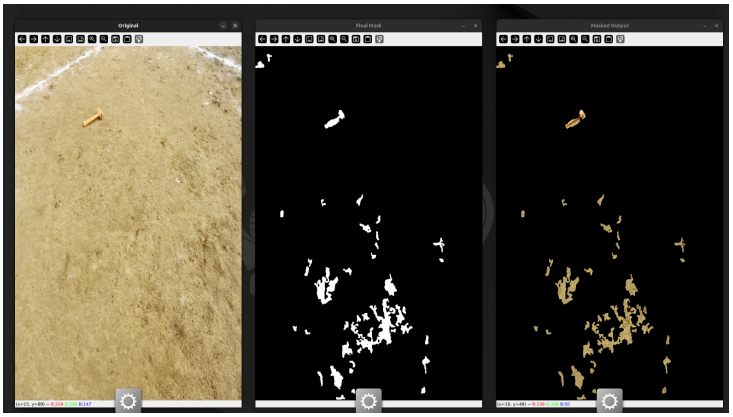
Using both binary and Otsu thresholding improved the robustness of mallet detection.

## 1.2 Failure Cases

1. Multiple objects with shadows

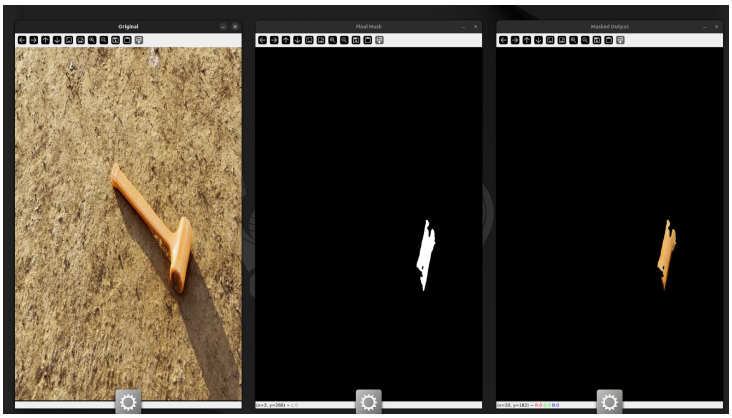


2. Sunlight making the background and mallet of similar hues



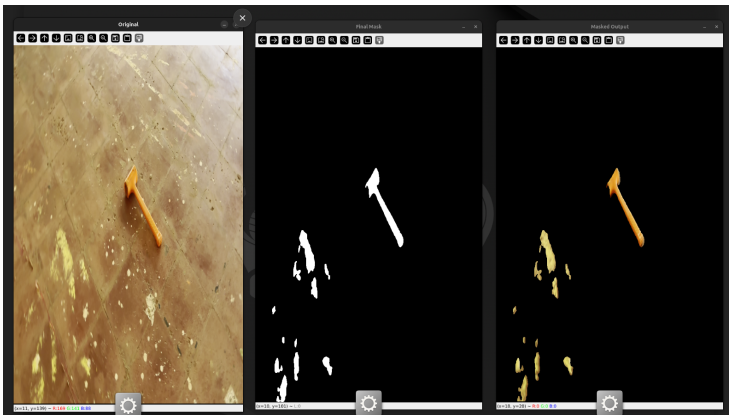
Morphological opening/closing could not resolve this without affecting other videos.

3. Mallet not fully masked at times




While removing background noise, part of the mallet may be masked.

4. Paint in background




Paint matching the mallet colour is also masked.

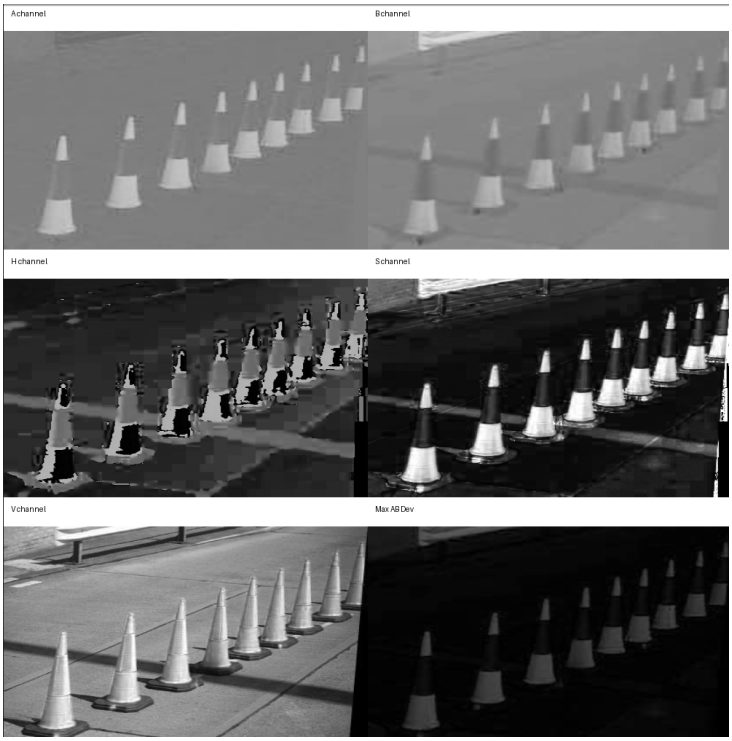
# 2 Exercise 2

## 2.1 Overview of Pipeline

Color alone is insufficient due to the presence of cones of multiple colors, illumination changes, and background similarity. Edge-based methods capture cone structure but may introduce background clutter. Using Hough lines initially produced too many lines, making processing infeasible. Instead, Canny edge detection with geometric analysis was used. Later, probabilistic Hough lines were applied to validate detected contours.

## 2.2 Colour Space Analysis

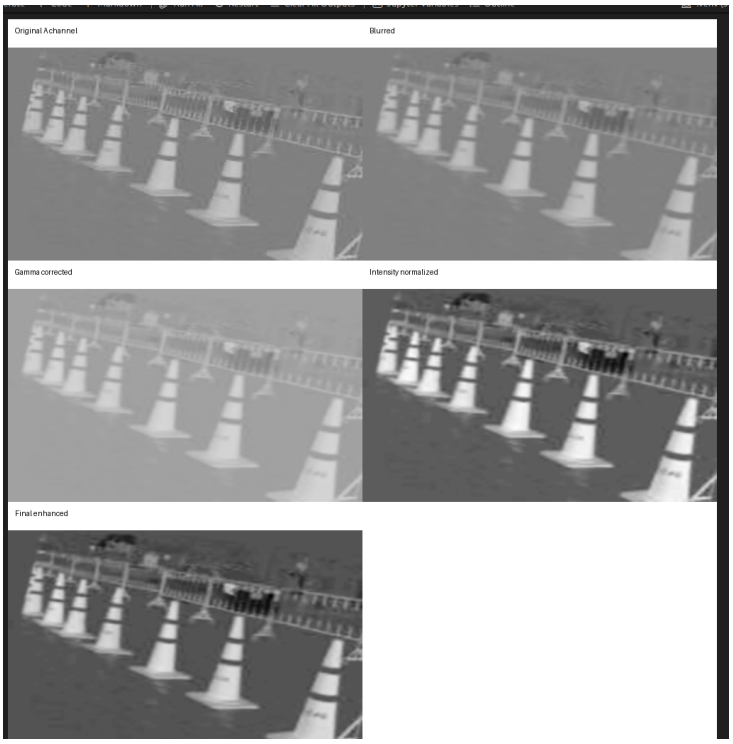
The A and B channels in LAB highlight deviations from neutral gray. Useful for detecting colorful regions while suppressing background. B could not separate cones from background noise, and A could not separate yellow and sometimes blue cones. Therefore, a combination was used: for each pixel, the maximum deviation from neutral in both channels was chosen.




## 2.3 Pre-processing

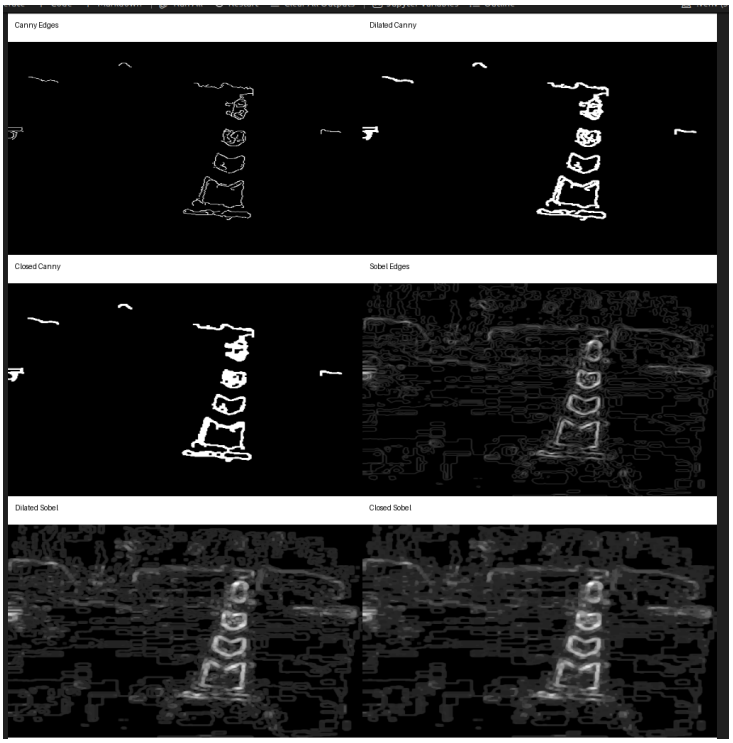
**Steps:**

1. Gaussian Blur to reduce background texture
2. Gamma Correction to enhance dark regions and control highlights
3. Intensity Normalisation to highlight bright cone colors
4. Pixel intensity scaling for contrast control




## 2.4 Edge Detection

Canny edge detection highlighted sharp cone edges while reducing background noise.




## 2.5 Contour Selection and Filtering

Wide contours were rejected as unlikely cones. Watershed segmentation separated overlapping cones.

```
contours, _ = cv2.findContours(
        edges_closed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )

contours = sorted(contours, key=contour_center_x)

filtered_contours = []

for cnt in contours:
    if cv2.contourArea(cnt) < 300:
        continue
    x, y, w, h = cv2.boundingRect(cnt)

    if h == 0:
        continue

    aspect_ratio = w / h   # width-to-height

    # Reject very wide contours that are unlikely to be cones
    if aspect_ratio > 2.5:
        continue

    filtered_contours.append(cnt)

# Attempt to split wide/merged contours using watershed
processed_contours = []
for cnt in filtered_contours:
    x, y, w, h = cv2.boundingRect(cnt)
    if w > max(1.3 * h, 120):
        splits = split_merged_contour(cnt, orig_img)
        processed_contours.extend(splits)
    else:
        processed_contours.append(cnt)

plot_images(frame_size=(500,300), **{'Original A channel': A, 'Detected Contours': cv2.drawContours(orig_img.copy(), proc
```


## 2.6 Grouping Contours

Contours with white stripes were grouped based on horizontal proximity. Geometric validation required bottom width $\geq 0.75\times$ top width and aspect ratio between 1.2 and 4.5.

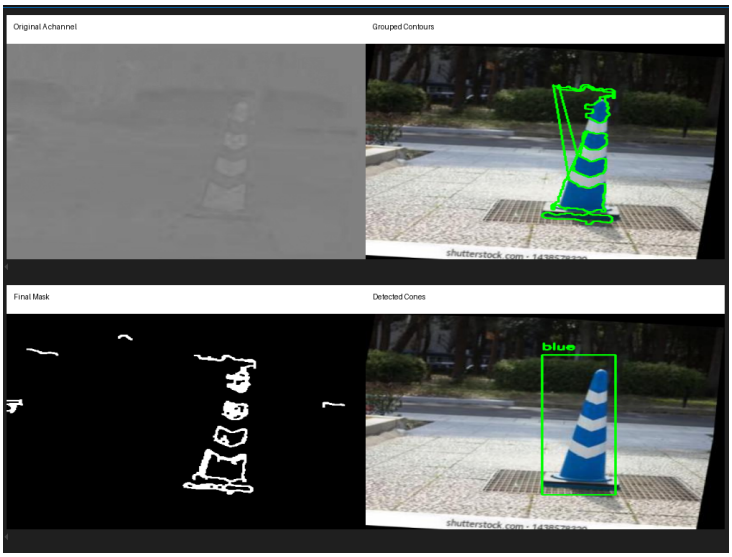
## 2.7 Use of Hough Lines

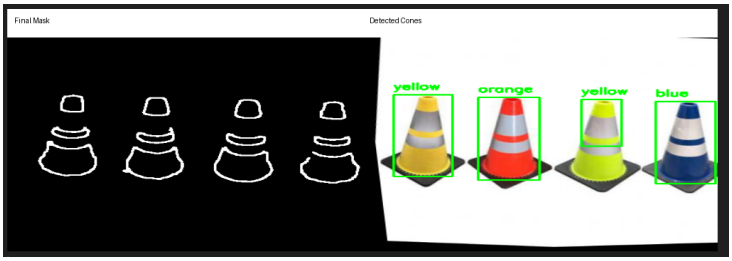
Probabilistic Hough lines validated cone contours. Only contours with $\geq 50\%$ slanted lines were classified as cones.

## 2.8 Colour Detection

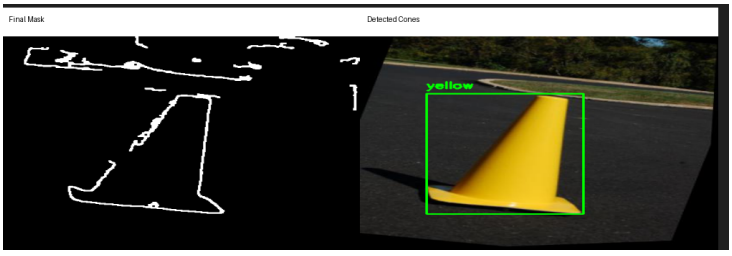
HSV color space was used. Ranges for H and S were defined for orange, yellow, blue, green, and pink cones. Majority color within the contour was assigned.

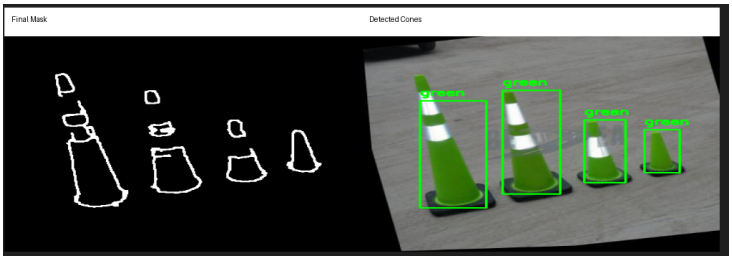
## 2.9 Successful Detections







## 2.10 Failure Cases

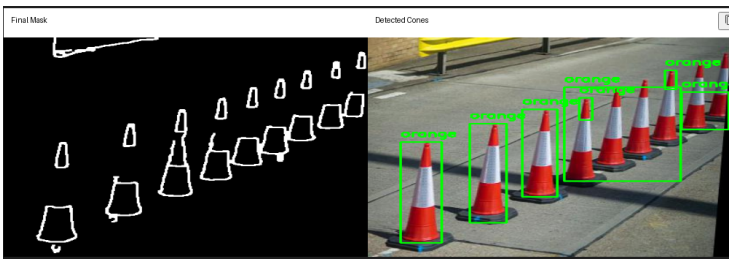



**Figure 4:** Multiple cones of same color overlap and are detected together.

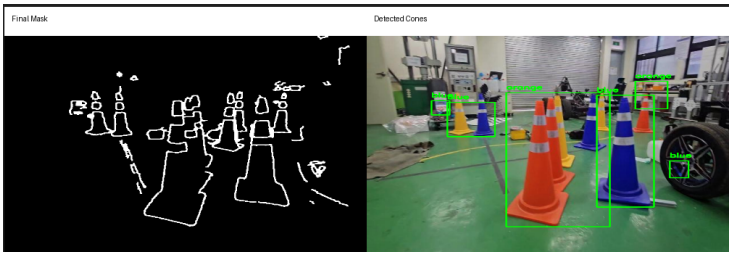



**Figure 5:** Overlapping cones create issues in contour overlap and separation.

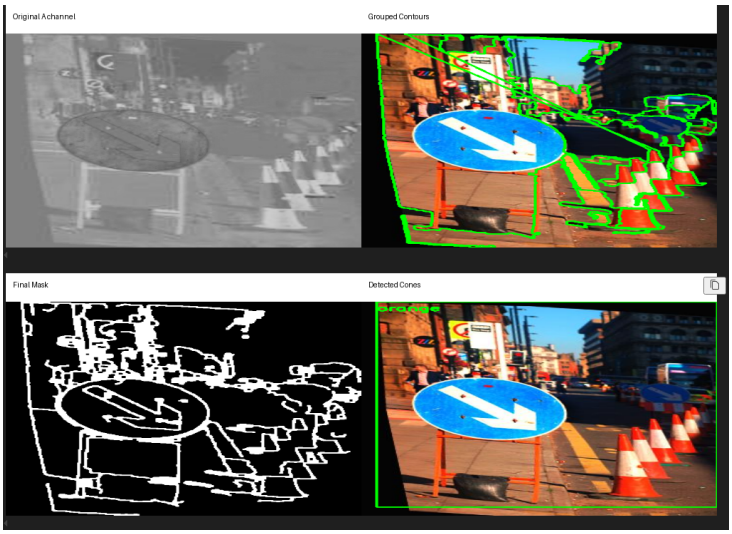



**Figure 6:** Too much background noise makes contour detection challenging.

# 3 Exercise 3

## 3.1 Overview

The inbuilt Aruco marker detection in OpenCV worked well, except for some false positives in background noise. Pre-processing using CLAHE, histogram equalization, bilateral filtering, and Gaussian blur reduced noise while maintaining edge clarity.

For small markers, blurring worsened detection. Filters were applied only if detected markers exceeded a size threshold.


**Figure 7:** Example of Aruco marker detection after pre-processing.