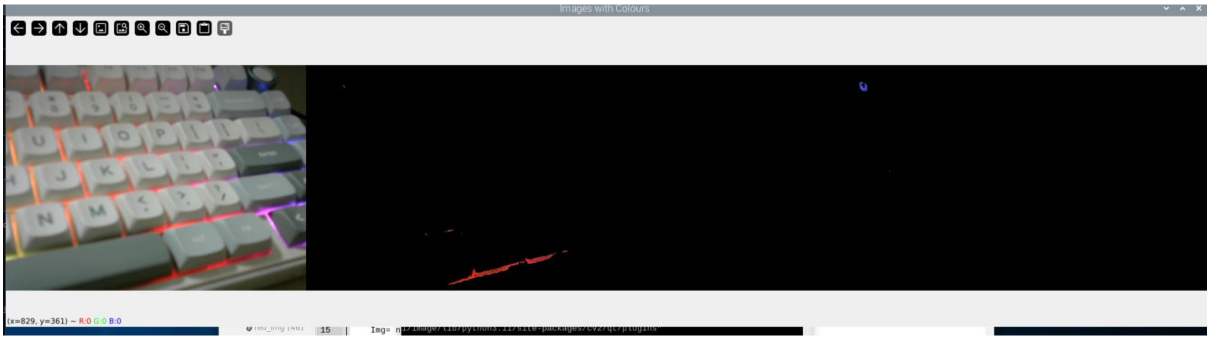
Image Analytic Lab

Real-time Image Processing with OpenCV

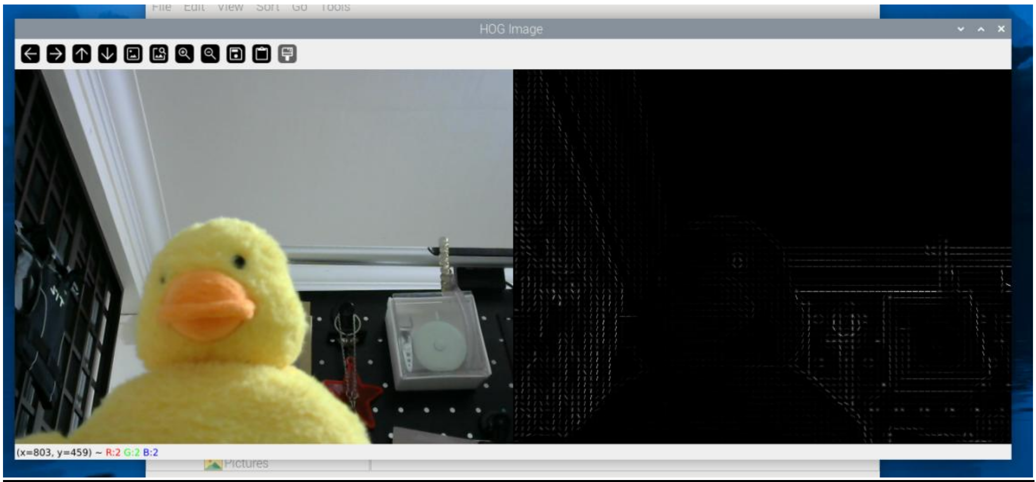
Using image_capture_display.py, read frames from the webcam and segment them into red, green and blue according to the intensity range for each colour channel (RGB).



Modify image_capture_display.py to segment frames into Yellow colour

```
hand_landmark.py    hand_landmark_modified.py    hand_gesture.py    obj_detection.p
# Refer to https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html
boundaries = [
        ([17, 15, 100], [50, 56, 200]), # For Red
        ([86, 31, 4], [220, 88, 50]), # For Blue
        ([25, 90, 4], [62, 200, 50]), # For Green
        ([20, 100, 100], [30, 255, 255]) # For Yellow
]

#%% Normalize the Image for display (Optional)
def normalizeImg (Img):
    Img= np.float64(Img) #Converting to float to avoid errors due to division
    norm_img = (Img - np.min(Img))/(np.max(Img) - np.min(Img))
    norm_img = np.uint8(norm_img*255.0)
    return norm_img

#%% Open CV Video Capture and frame analysis
cap = cv2.VideoCapture(0)

# Check if the webcam is opened correctly
if not cap.isOpened():
    raise IOError("Cannot open webcam")

# The loop will break on pressing the 'q' key
while True:
    try:
        # Capture one frame
        ret, frame = cap.read()

        output=[]

        # loop over the boundaries
        for (lower, upper) in boundaries:
            # create NumPy arrays from the boundaries
            lower = np.array(lower, dtype = "uint8")
            upper = np.array(upper, dtype = "uint8")

          # find the colors within the specified boundaries and apply the mask (basically segme
            mask = cv2.inRange(frame, lower, upper)
            output.append(cv2.bitwise_and(frame, frame, mask = mask)) #Segmented frames are a

        # Output is appeneded to be of size Pixels X 3 (for R, G, B)
        red_img = normalizeImg(output[0])
        green_img = normalizeImg(output[1])
        blue_img = normalizeImg(output[2])
        yellow_img = normalizeImg(output[3])

        # horizontal Concatination for displaying the images and colour segmentations
        catImg = cv2.hconcat([frame,red_img,green_img,blue_img, yellow_img])
        cv2.imshow("Images with Colours",catImg)

        if cv2.waitKey(1) & 0xFF == ord('q'):
```

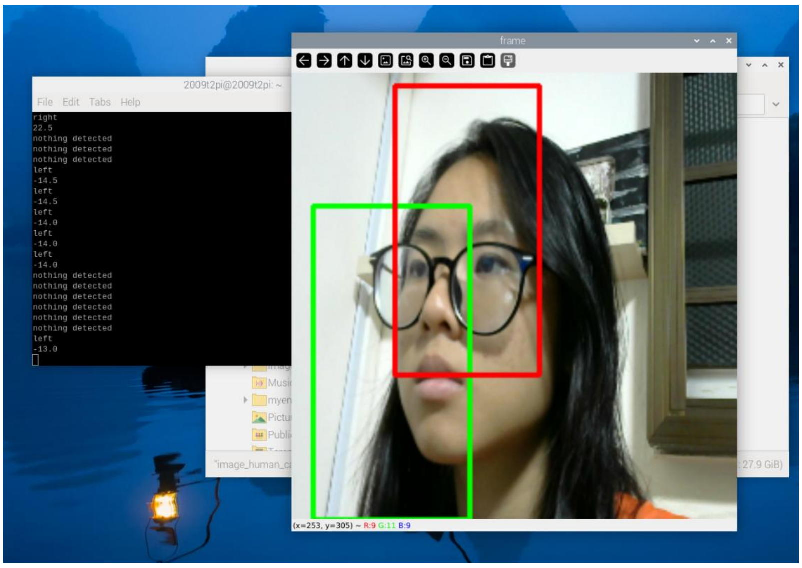
Real-time Image Analysis with Scikit-Image Library



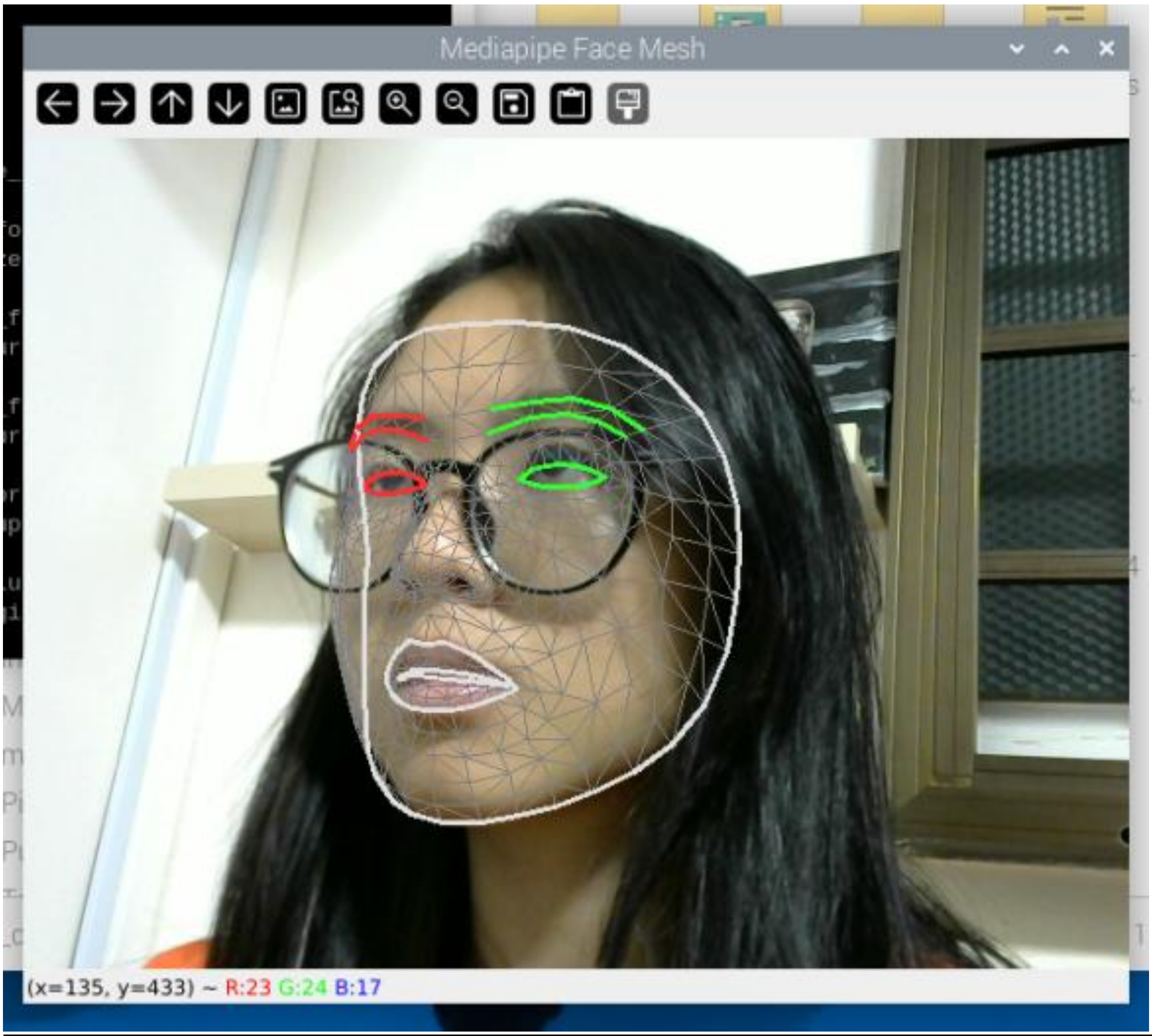
After downsizing the image, frame rate is faster & quicker to extract HoG feature

Detect human in a live webcam feed with OpenCV's HoG Descriptor

Also determines relative position of detected person & prints whether they are cantered.



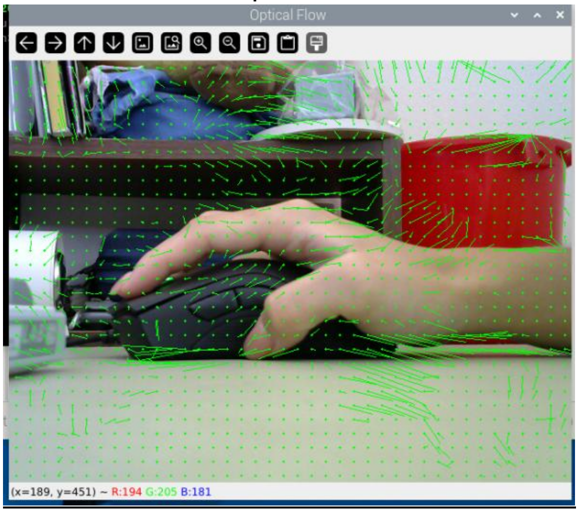
Real-time Image Feature Analysis for Face Capture & Facial Landmark Extraction with Mediapipe



Video Analytic Lab

Real-time Video Processing with Optical Flow

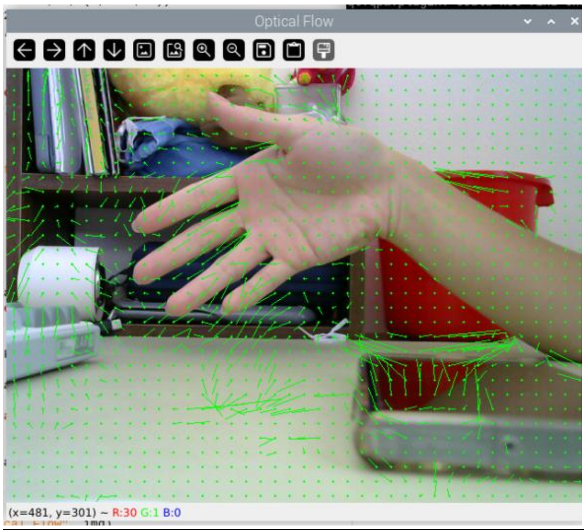
Flow Farneback Optical Flow



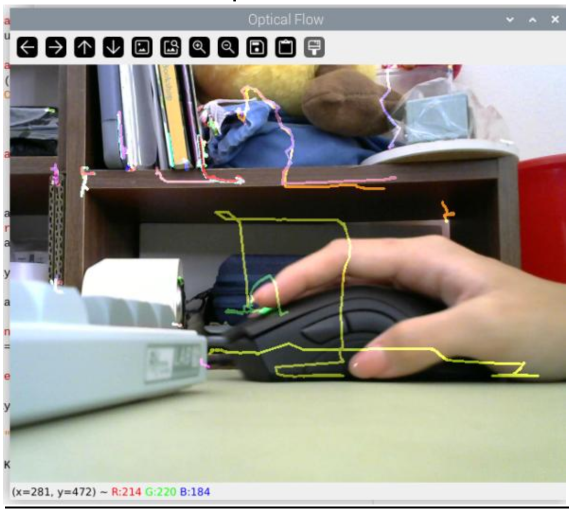
Farneback Optical Flow estimates dense motion across an image by analysing pixel-level changes. The window size determines the area around each pixel used for motion estimation, with larger values improving robustness but potentially adding noise.

Pyramid levels create downscaled versions of the image to track fast movements, where higher values improve large motion tracking but increase computation.

Stopping criteria define when the algorithm stops refining motion estimates, balancing accuracy and speed. The more iterations yield better results but take longer, while higher epsilon values stop earlier and may miss small refinements.



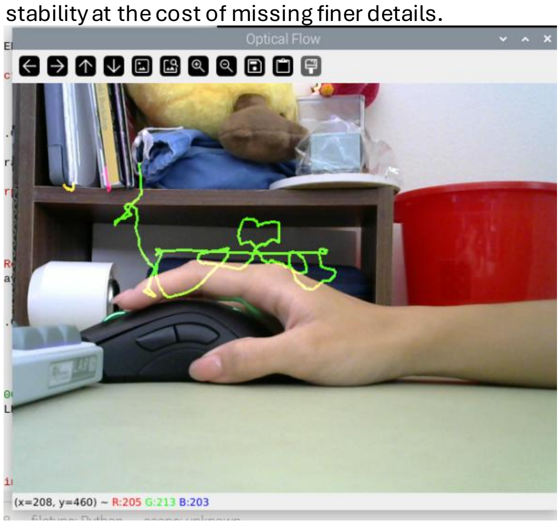
Lucas Kanade Optical Flow



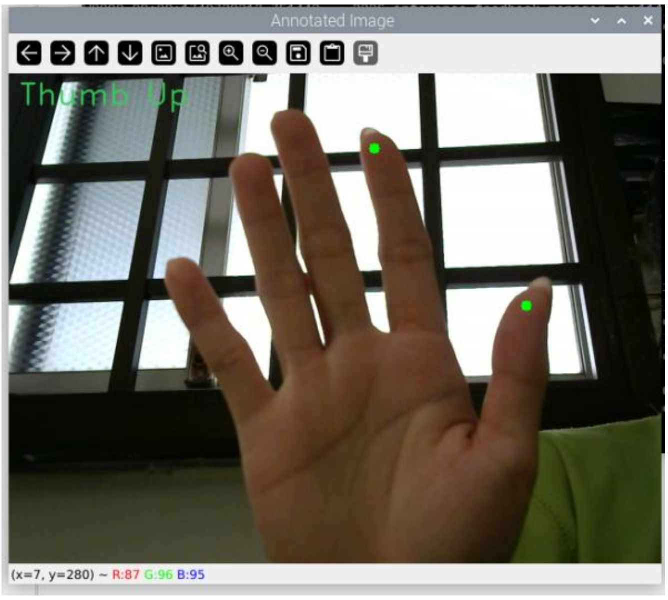
Lucas-Kanade Optical Flow detects and tracks key feature points rather than analysing every pixel. The maximum number of keypoints affects how many strong features are detected, with higher values improving tracking accuracy.

The minimum corner strength determines which keypoints are kept based on quality, where higher values ensure only strong features are used. The minimum distance between corners controls how close keypoints can be, with larger distances reducing overlap.

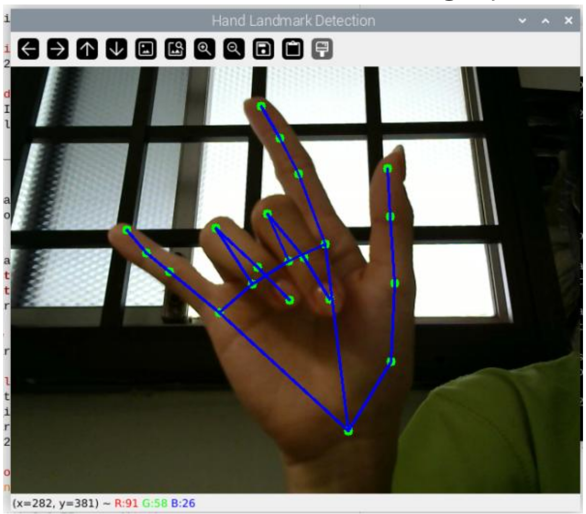
Lastly, the block size defines the region around each feature used for detection, with larger values improving stability at the cost of missing finer details.



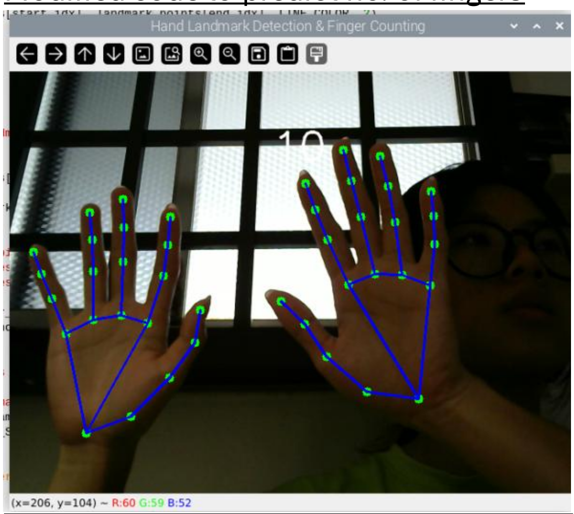
Advanced Video Analytics with Mediapipe and handlandmark detection model



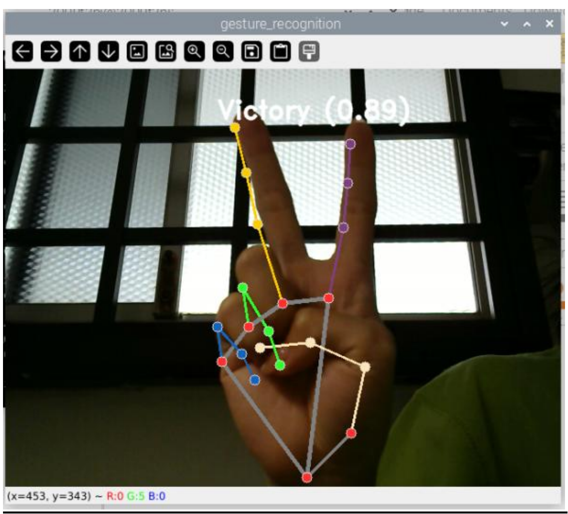
Modified code to show all 21 finger points

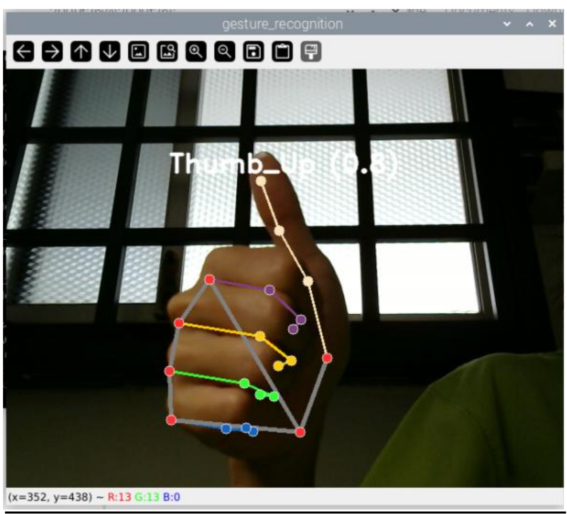


Modified code to predict no. of fingers



Advanced Hand Gesture Recognition





Advanced Object Detection