# Vision for HCI

## Description

In this session, you will learn how to use machine vision to make new controls for a game.
This worksheet will guide you through the different steps to make a basic hand gesture controller. But you are free to spice up things and explore other types of interactions.

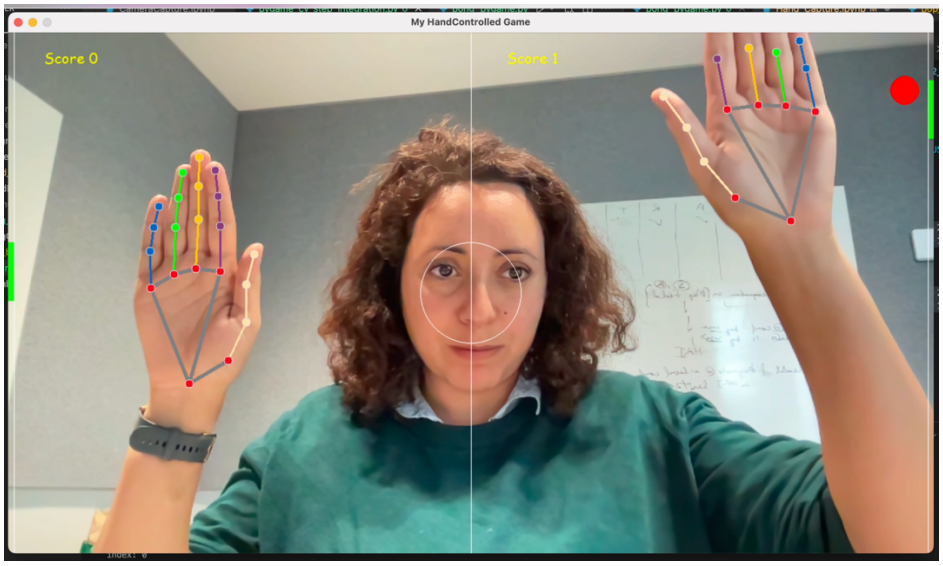



*Figure 1| Screenshot of the pong game in which the green paddles are controlled by the hands. In this example, we use the y coordinate of the index's root (Metacarpi - MCP) joint position to set the y-position of the paddles.*

## Objectives

- To understand the concept of particle systems and implement it
- To integrate a gesture/object recognition system into a game as a controller

## Preliminary steps

To complete this workshop, you will need a computer with a webcam. You will also need to know Python and have installed mediapipe as well as pygame.

We recommend you to setup a virtual environment for this:

```
$ python3 -m venv mp_env && source mp_env/bin/activate
```

You can then install mediapipe

```
$ pip install mediapipe
$ pip install pygame
```

## Part 1 Getting Familiar with OpenCV and MediaPipe

The first part is to get familiar with OpenCV and mediapipe.
In this part, you will learn how to read and display images and use mediapipe to detect hands on the images.

## 1.1 Loading an Image and Displaying it using OpenCV

Take a picture of yourself with a thumbup
We will be using this image to get familiar with the hand detection using mediapipe

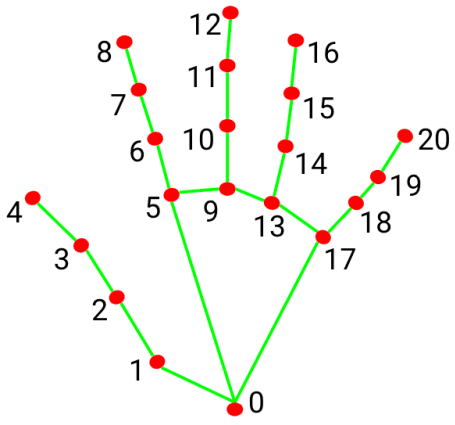
Use the code below to read and display the image using OpenCV.
OpenCV is a big library, and you can find documentation here:
https://opencv.org/

```
import cv2

image = cv2.imread("thumbup.jpeg")
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(1) # normally unnecessary, but it fixes a bug on MacOS where the window doesn't
close view
```

## 1.2 Recognizing and displaying the Hands

To get the hands information (number of hands, position and pose of the joints of the hands), we use mediapipe.
Mediapipe is a set of pre-trained model that have been wrapped up some nice APIs available in different programming language.
The model of the hand in mediapipe:




0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

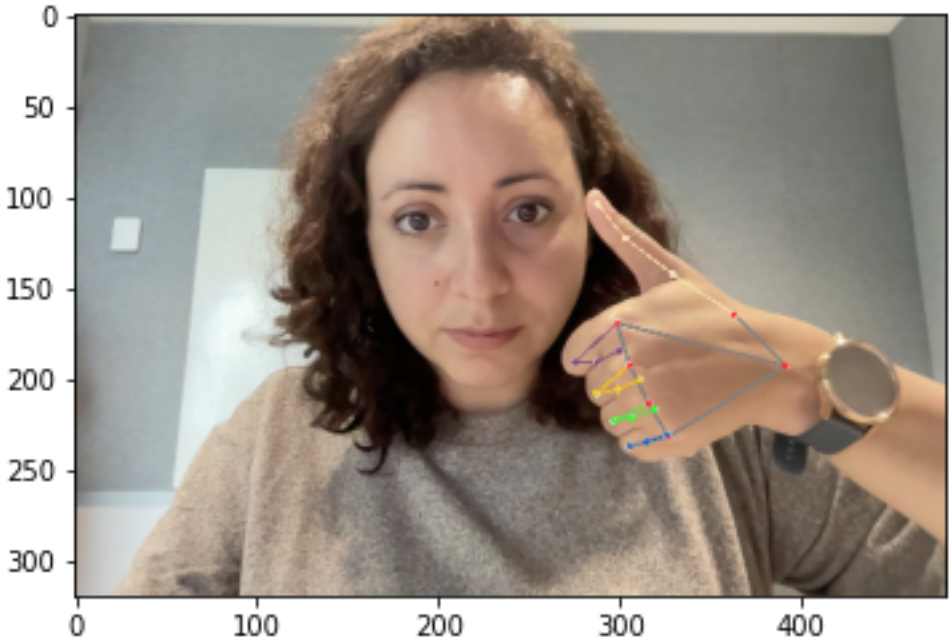
### 1.2.1 Loading the model:

In a new Python file, run the below code.
Read the documentation the Hands class to understand its structure.

```
import mediapipe as mp
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
help(mp_hands.Hands)
```

### 1.2.2 Running Medipipe Hands in your image

Now modify your code that reads an image with opencv to display the hand's information. You can use the code-snipped below.




Try the code with different images and observe the console output. What changes? What are the values outputted?

```
# Run MediaPipe Hands.
with mp_hands.Hands(
    static_image_mode=True,
    max_num_hands=2,
    min_detection_confidence=0.7) as hands:

    # Convert the BGR image to RGB, flip the image around y-axis for correct
    # handedness output and process it with MediaPipe Hands.
    results = hands.process(cv2.flip(cv2.cvtColor(image, cv2.COLOR_BGR2RGB), 1))

    # Print handedness (left v.s. right hand).
    print(results.multi_handedness)

    # Draw hand landmarks of each hand.
    image_hight, image_width, _ = image.shape
    annotated_image = cv2.flip(image.copy(), 1)
    for hand_landmarks in results.multi_hand_landmarks:
      # Print index finger tip coordinates.
      print(
          f'Index finger tip coordinate: (',
          f'{hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].x *
image_width}, '
          f'{hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y *
image_hight})'
      )
      mp_drawing.draw_landmarks(
          annotated_image,
          hand_landmarks,
          mp_hands.HAND_CONNECTIONS,
          mp_drawing_styles.get_default_hand_landmarks_style(),
          mp_drawing_styles.get_default_hand_connections_style())
    resize_and_show(cv2.flip(annotated_image, 1))
```

## 1.3 Using your Webcam with VideoCapture

Now, we want our game to work on real-time capture image. So we will now try to stream our video camera stream and displaying it using opencv. Below is the code snippet that lets you read and stream your webcam images.

```
import cv2

cap = cv2.VideoCapture(0)
cap.set(3,640) # adjust width
cap.set(4,480) # adjust height

while True:
    success, img = cap.read()
    cv2.imshow("Webcam", img) # This will open an independent window
    if cv2.waitKey(1) & 0xFF==ord('q'): # quit when 'q' is pressed
        cap.release()
        break
cv2.destroyAllWindows()
cv2.waitKey(1)
```

## Part 2 Simple Hand Controller using Mediapipe

In this part, to get familiar with the hand information, you will implement a small program that will read the hands information and control the threshold filter of the image.(see https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html )

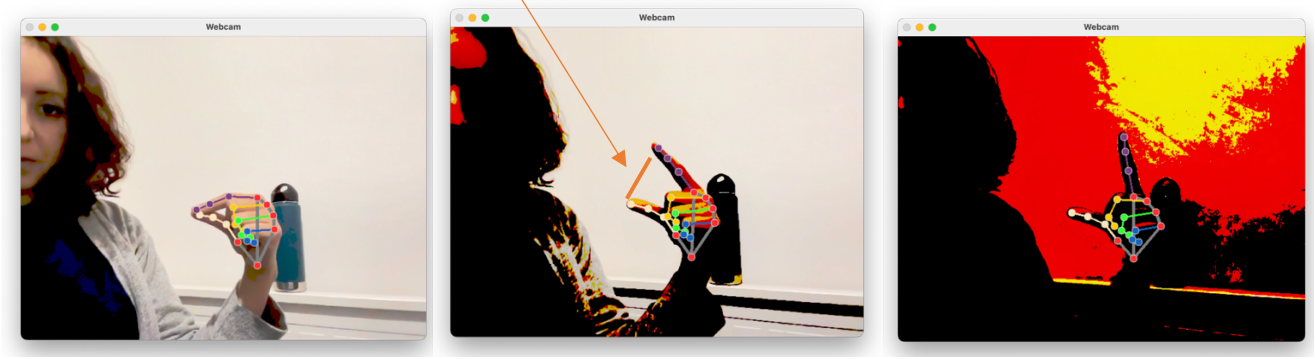
The threshold function below, takes the pixels in the images and basically set their value to zeros if they are in the range of intensity between 50 and 120.

```
ret, thresh = cv2.threshold(annotated_image, 50, 120, cv2.THRESH_TOZERO)
```

Now what we want is to use hand gesture to tune this range in real-time.

```
ret, thresh = cv2.threshold(annotated_image, 50+125/100*int(eDistance),
120+125/100*int(eDistance), cv2.THRESH_TOZERO)
```

For that we will use the distance between the tip of the thumb and the index (eDistance)
Write the code to compute **eDistance**.



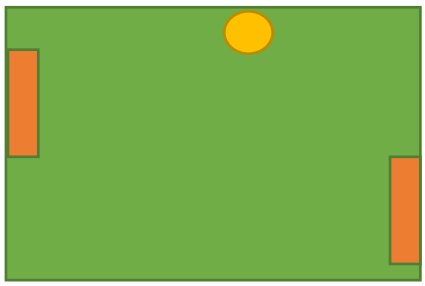

Note that we use a scaling factor (`125/100`) that might depend on the size of your image and the distance of your hands to the screen. Feel free to adjust it to have a nice and wide range of output image as you bring your thumb and index together or apart.

# Part 3 The Hand-Controlled Pong Game

In this part, we will see how to setup the pong game and how to integrate the hand controller.

## 3.1 Pong Game Explained

The game of pong is fairly basic. We have a ball, bouncing on the top and bottom part of our game arena. When the ball bounces on a paddle (placed on the left and right side of the arena) it is saved, if the ball passes through and touches the left of right side of the arena, a point is given to the player at the opposite side.



*Ball bouncing on the top border of the arena*

*Ball bouncing on the right paddle*

*Ball touching the right side of the arena, resulting in one point attributed to the left player's score*

## 3.2 PyGame Basics

Pygame is a python library that allow to easily create games. It has building functions to draw elements and manage events (such as key events0 that could be used to control our game.
But for us it will be mainly used to setup the graphics and to handle the frame rate.

You can already open the documentations of pygame and see the different functions to draw the elements of our game.
https://www.pygame.org/docs/ref/draw.html

## 3.3 Setting-up the Game

We provide to you the skeleton of code for a game using pygame.

Run the code and observe.

```
import pygame


# Initialize
pygame.init()

# Create Window/Display
width, height = 1280, 720
window = pygame.display.set_mode((width, height))
pygame.display.set_caption("My Hand Controlled Game")

# Initialize Clock for FPS
fps = 30
clock = pygame.time.Clock()


# Main loop
start = True
while start:
    # Get Events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            start = False
            pygame.quit()

      # Game Logics….


    # Update Display
    pygame.display.update()
    # Set FPS
    clock.tick(fps)
```

## 3.4 Draw the paddles and the ball

The Paddles
Using the rect function of pygame, draw the left and right paddles

```
PAD_WIDTH = 8
PAD_HEIGHT = 80
```

Draw the ball using the circle function

3.5 Game physics

```
# Main loop
start = True
while start:
    # Get Events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            start = False
            pygame.quit()
             #########
             ## Capture the index position
```

```
      # update paddle's vertical position, keep paddle on the screen
    if l_paddle_pos[1] > HALF_PAD_HEIGHT and l_paddle_pos[1] < height - HALF_PAD_HEIGHT:
        l_paddle_pos[1] += l_paddle_vel
    elif l_paddle_pos[1] == HALF_PAD_HEIGHT and l_paddle_vel > 0:
        l_paddle_pos[1] += l_paddle_vel
    elif l_paddle_pos[1] == height - HALF_PAD_HEIGHT and l_paddle_vel < 0:
        l_paddle_pos[1] += l_paddle_vel

    if r_paddle_pos[1] > HALF_PAD_HEIGHT and r_paddle_pos[1] < height - HALF_PAD_HEIGHT:
        r_paddle_pos[1] += r_paddle_vel
    elif r_paddle_pos[1] == HALF_PAD_HEIGHT and r_paddle_vel > 0:
        r_paddle_pos[1] += r_paddle_vel
    elif r_paddle_pos[1] == height - HALF_PAD_HEIGHT and r_paddle_vel < 0:
        r_paddle_pos[1] += r_paddle_vel

    #update ball
    ball_pos[0] += int(ball_vel[0])
    ball_pos[1] += int(ball_vel[1])
```

```
   #ball collision check on top and bottom walls
    if int(ball_pos[1]) <= BALL_RADIUS:
        ball_vel[1] = - ball_vel[1]
    if int(ball_pos[1]) >= height + 1 - BALL_RADIUS:
        ball_vel[1] = -ball_vel[1]

    #ball collison check on gutters or paddles  == check for the left paddle
    if int(ball_pos[0]) <= BALL_RADIUS + PAD_WIDTH and int(ball_pos[1]) in range(l_paddle_pos[1] -
HALF_PAD_HEIGHT,l_paddle_pos[1] + HALF_PAD_HEIGHT,1):
        ball_vel[0] = -ball_vel[0]
        ball_vel[0] *= 1.1
        ball_vel[1] *= 1.1
    elif int(ball_pos[0]) <= BALL_RADIUS + PAD_WIDTH:
        r_score += 1
        ball_init(True)
       ##### WRITE the code for the right paddle

    #draw paddles and ball
   # ….
```

## 3.6 Convert CV image into canvas

Now that you have the game setup, you will need to add you code to stream the video using opencv (as per section 1.3 and use mediapipe to obtain the position of the base of the index (this is the join we use to control the paddle).
The code below show you how to convert an opencv image into a surface array that can be use as a background of our game.

```
imgRGB = cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB)
imgRGB = np.rot90(imgRGB)
frame = pygame.surfarray.make_surface(imgRGB).convert()
frame = pygame.transform.flip(frame, True, False)
window.blit(frame, (0, 0))
```

Note that you will need to capture the position of two hands, you can easily associate the control of the left and right paddle by using the x-position of the hand on the image. (i.e. if x<width/2 => control of the left paddle, else => control of the right paddle)

## Bonus: Use the whole hand

In the game so far, we have been using only the y position of the base of the middle finder to control the position of the paddle.

In this bonus, we want to make the more fun and to actually use the whole hand as a pong racket.
To do that, we would need to compute the collision between the hands and the ball.
One way to do that is to consider a rectangular bounding box around the hand.
The collision detection then just comes back to detecting rather if the ball touches a line of this bounding box.

## Bonus+: Rock-Paper-Scissors using Hands (Training a ML model)

In this session, you saw how you could use MediaPipe to write a small pong game.
Although fun, it is not necessarily more natural to play pong with your hands compared with the keyboard.

In this bonus, I propose that you try to implement a rock-paper-scissors game to play against the computer using the hand-recognition.

To do that you will need several modules:

1. the game logic: here the idea can be to attribute different distributions for the computer to pick between rock, paper and scissors. You could for instance bias it towards rock and decide that 40\% of the times, the computer will play rock, 35\% for scissors and 25\% for paper.
2. classification between the three gestures. For that have a look at this link that will explain you how to use mediapipe to train a hand gesture classifier.

   This tutorial can guide you in training a new gesture model with the three classes (rock, paper, Scissors) : https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/

   Note that for that you will need to record some samples of the gestures
   You will also need to install tensorflow to train the classification model.