Explanation of Codes:

1. For the preparation, we need to import OpenCV and MediaPipe to use their functions. In addition, import time to give a fixed duration of the display of “Wave” to make the result easier to see, or else it will be displaced by the text of other gestures immediately. After that, we set up the video capture feature and initiated a video writer to save the result as an mp4 video, similar to what we did in lab03 but the format of the video is different.

```
import cv2
import mediapipe as mp
import time

cap = cv2.VideoCapture(0)

result = cv2.VideoWriter('result.mp4',
cv2.VideoWriter_fourcc('m', 'p', '4', 'v'),
15, (1068,600),4)
```

2. We set up the variables that need to be used in later steps. For example, we create the hand object in order to make the functions of MediaPipe work. We only need to simply get a frame from the video to measure the shape before any of the loops start. Because the height and width of the given video are fixed, we don’t need to repeat this step which will cost extra time efficiency during the loops.

   The lists that record x and y coordinates are used in building rectangles and gesture detections.
   (Tips are for all three gestures.
   The wrists and time features are for wave.
   Pips are for fist, great thumb, and number gestures.)
   The hand center coordinates and the other variables are used in drawing track lines.

```
hands = mp.solutions.hands.Hands()
exist, image = cap.read()
height, width, channels = image.shape
x_coordinates =[]
y_coordinates =[]
previous_wrist_x = 0
wrist_x = 0
previous_tips_x =[0,0,0,0,0]
tips_x =[]
tips_y =[]
pips_x =[]
pips_y =[]
pips_coordinates =[]
hand_center_x = 0
hand_center_y = 0
centers = []
i = 0
text = ""
wave_start = time.time()
wave_end = time.time()
```

3. Start the main while loop to ensure the video capture process will not stop until the given video ends. After that, get frames to detect and track hands. Because MediaPipe requires RGB images to get the status of the hand landmarks, we need to convert the frame we get into the RGB scale at first. I also used “try…catch…” feature to prevent the error of the resulting video saving when the given video reaches its end. If this problem is not prevented, the resulting video can be unplayable.

```
while cap.isOpened():

    try:
        exist, image = cap.read()
        colored_image = cv2.cvtColor(image, 4)
        hand_lms_exist = hands.process(colored_image).multi_hand_landmarks

    except Exception:
        print("The video has ended, so the window is closed.")
        break
```

4. If the status of hand landmarks is normal, we use two loops to check every landmark of each hand. Then we get the x and y coordinates of each landmark, and add them to the list we have created. When the inner loop ends, we use the built-in method of list: max() and min() to get the starting point (point with the smallest x and y value) and ending point (point with the greatest x and y value). Finally, we use those two points to draw the two green rectangles. I increased the size of the rectangles a bit because if the rectangles use exactly two points, they can not cover the fingers very well.

```
if hand_lms_exist:
    for handLMs in hand_lms_exist:
        for lm in handLMs.landmark:

            x_coordinates.append(int(lm.x * width) )
            y_coordinates.append(int(lm.y * height) )

        start_x = min(x_coordinates)
        start_y = min(y_coordinates)
        end_x = max(x_coordinates)
        end_y = max(y_coordinates)
        cv2.rectangle(image, (start_x - 10, start_y - 10),
        (end_x + 10, end_y + 10), (0, 255, 0), 2)
```

5. After we finished drawing the two rectangles during each outer loop, we reused the coordinates of the starting points and ending points. By adding them together and dividing them by 2, we can get the coordinate of the centers of each rectangle, which can be used to draw the tracking lines of the hand motion. After

adding the center coordinates to the list we have created, we will draw the blue tracking lines according to the extracted x and y values (x1, y1, x2, y2) between points. After each time the outer loop ends, set the variables to make them reusable in the next loop.

There are three important things to mention.

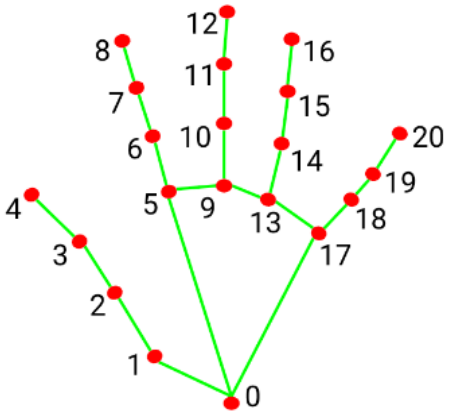
First, I give an upper bound of 100 to the x and y coordinates of the track line points. This is to avoid a sudden jump of the line when the second hand appears in the video or when the second hand disappears from the video.

The second thing is that because the points are no longer adjacent to each other after the second hand appears, I choose to compare points “i” and point “i+2”. This is to avoid the situation that the two points on each hand are linked together instead of demonstrating the track lines of the hand motions.

The last thing is that because each frame will erase the single line drawn during the previous frame, we have to build a loop and use an additional list to record and repeat all lines that we have drawn before. Else we can not get the proper result.

```
hand_center_x = round((start_x + end_x)/2)
hand_center_y = round((start_y + end_y)/2)
centers.append((hand_center_x, hand_center_y))
while(i+2<len(centers)):
    x1, y1 = centers[i]
    x2, y2 = centers[i+2]
    a = x1-x2
    b = y1-y2
    if (abs(b)>100 or abs(a)>100):
            i += 1
            pass
    else:
            cv2.line(image, (x1, y1), (x2, y2), (255, 0, 0), 4,1)
            i += 1
i = 1
```

6. Here is the process to get the x and y coordinates of the tips and pis. According to the landmark figure in Github:



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

Fig 2. 21 hand landmarks.

The five tips have indexes of 4, 8, 12, 16, and 20 in the total landmark array.
The five pips have indexes of 2, 6, 10, 14, and 18 in the total landmark array.
The wrist's index is always 0 in the total landmark array.

```
while(i<6):
    tips_x.append(x_coordinates[4*i])
    tips_y.append(y_coordinates[4*i])
    pips_x.append(x_coordinates[4*i-2])
    pips_y.append(y_coordinates[4*i-2])
    i += 1
i = 0
wrist_x = x_coordinates[0]
```

7. Feature to define and detect waves. If the horizontal movements of all five fingertips are more evident than the horizontal movement of the wrist, then we can consider this gesture as a wave. I also add a threshold of 10 to avoid giving false positive values of the detection when unintentional shaking of hands happens.

```
waving = True
while(i<5):
    if (abs(tips_x[i] - previous_tips_x[i]) <=
    abs(wrist_x - previous_wrist_x) + 10):
          waving = False
    i += 1
i = 1
```

8. Feature to define and detect fist. If the thumb is facing towards the center of the hand, and all other fingers are facing downwards, then we can consider this gesture as a fist.

```
fisting = True
while(i<5):
    if (tips_y[i] < pips_y[i] or
    (tips_x[0] > pips_x[1] and tips_x[0] > pips_x[4] ) or
    (tips_x[0] < pips_x[1] and tips_x[0] < pips_x[4] )):
            fisting = False
    i += 1
i = 0
```

9. Feature to define and detect numbers. All number gestures between 1 to 5 require the index finger to face upward. If it is not, then this gesture is not a number expression. After that, check which fingers are down to specify the exact number represented by the gesture. For example, “2” requires the ring finger to face down, and “3” requires the pinky to face down.

```
number = True
value = ""
if (tips_y[1] > pips_y[1]):
        number = False
elif (tips_y[2] > pips_y[2]):
        value = "1"
elif (tips_y[3] > pips_y[3]):
        value = "2"
elif (tips_y[4] > pips_y[4]):
        value = "3"
elif ( (tips_x[0] < pips_x[1] and tips_x[0] > pips_x[4] ) or
    (tips_x[0] > pips_x[1] and tips_x[0] < pips_x[4] )):
        value = "4"
else:
        value = "5"
```

10. Identify the category of the gesture according to the above features. If all four fingers except the thump are facing downwards, and the thumb is facing outward of the hand, then we can consider this gesture as a thumb.

    Another thing worth mentioning is that the text “wave” is hard to see because it is not a static gesture. It will be overwritten by the text of the number gesture “5” once the hand stops moving. So I implemented the time library and gave it a period of 1 second before changing. This can make the text easier to be observed.

```
if (waving):
            wave_start = time.time()
            text = "Wave"
elif (number):
            wave_end = time.time()
            if (wave_end - wave_start >= 0.5):
                        text = "Number: " + value
elif (fisting):
            text = "Fist"
else:
            wave_end = time.time()
            if (wave_end - wave_start >= 0.5):
                        text = "Great thumb"
```

11. Output the text in colour yellow, font Italic, on top of the hand detection rectangles. Send the current landmark coordinates to the previous landmark coordinates, which will be used in the wave detections later. Reset all necessary variables so that we can use them in the next frame.

```
cv2.putText(image,text, (start_x, start_y - 20),
cv2.FONT_ITALIC, 1, (0,255,255),2)
while(i<5):
    previous_tips_x[i] = tips_x[i]
    i += 1
i = 0
previous_wrist_x = wrist_x
x_coordinates =[]
y_coordinates =[]
tips_x =[]
tips_y =[]
pips_x =[]
pips_y =[]
```

12. Show the result by using a window, resizing the frames and saving the resulting video to disk. The main loop will end if the captured video ends or the user hits the “space” key. After the main while loop ends, both the captured video and the saved video will be released.

```
    cv2.imshow("Hand Detection and Motion Tracking", image)
    result_image = cv2.resize(image,(1068,600))
    result.write(result_image)
    out = cv2.waitKey(1)
    if out == 32:
        break

cap.release()
result.release()
```