Fingertip trajectory recognition

1. Content Description

This course implements color image acquisition and fingertip detection using the MediaPipe framework. Gestures are used to start and stop recording fingertip trajectories within the image. After recording, a fingertip trajectory map is generated and the trajectory is recognized. This section requires entering commands in a terminal. The terminal you open depends on the board type. This lesson uses the Raspberry Pi as an example.

For Raspberry Pi and Jetson-Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [4.Enter Docker (For JETSON Nano and RPi 5)]**.

For RDKX5 and Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

2. Program Startup

For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

Enter the Docker container (see [Docker course] --- [4. Docker Startup Script] for steps).

All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

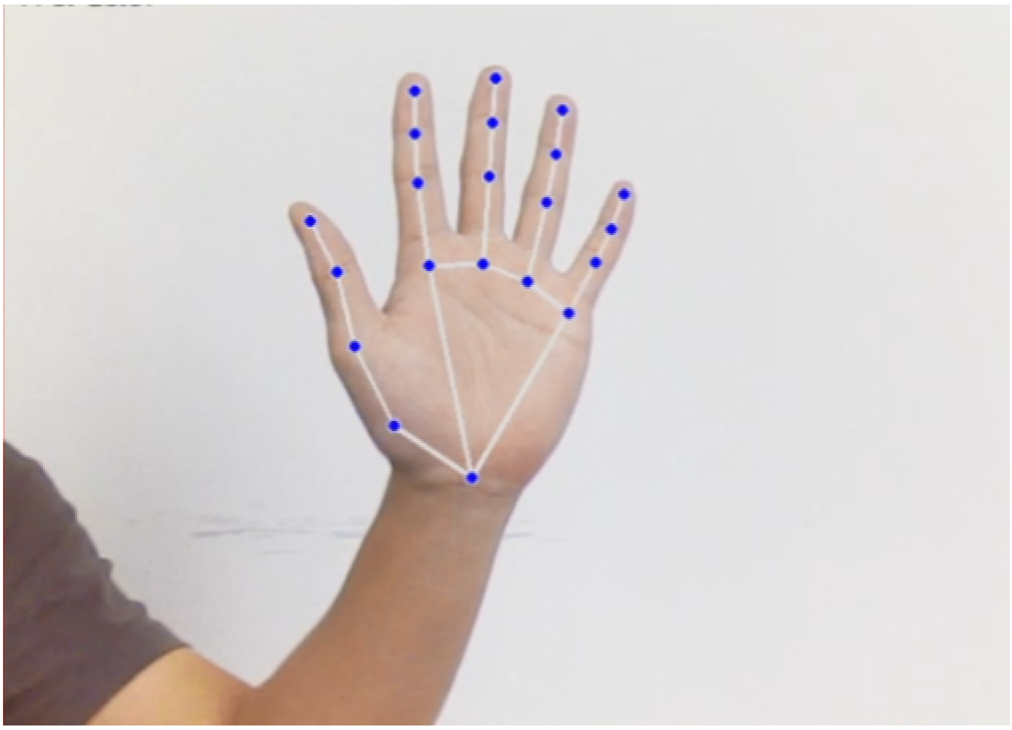
First, enter the following command in the terminal to start the camera:

```
#usb camera
ros2 launch usb_cam camera.launch.py
#nuwa camera
ros2 launch ascamera hp60c.launch.py
```

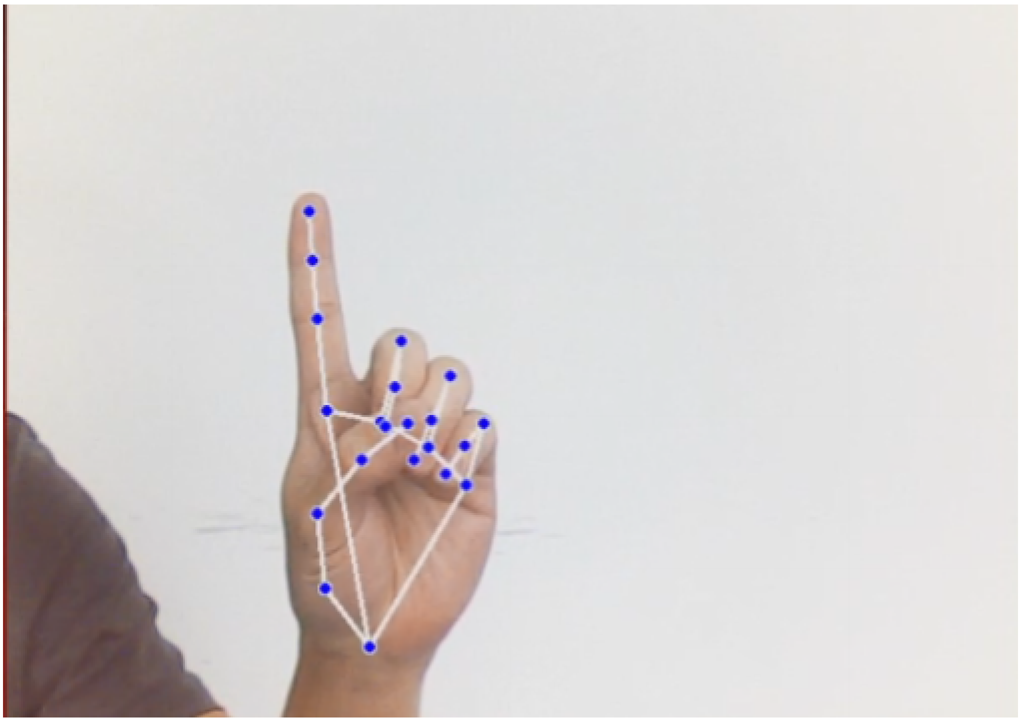
After successfully starting the camera, open another terminal and enter the following command to start the fingertip trajectory recognition program:

```
ros2 run yahboomcar_mediapipe 12_FingerTrajectory
```

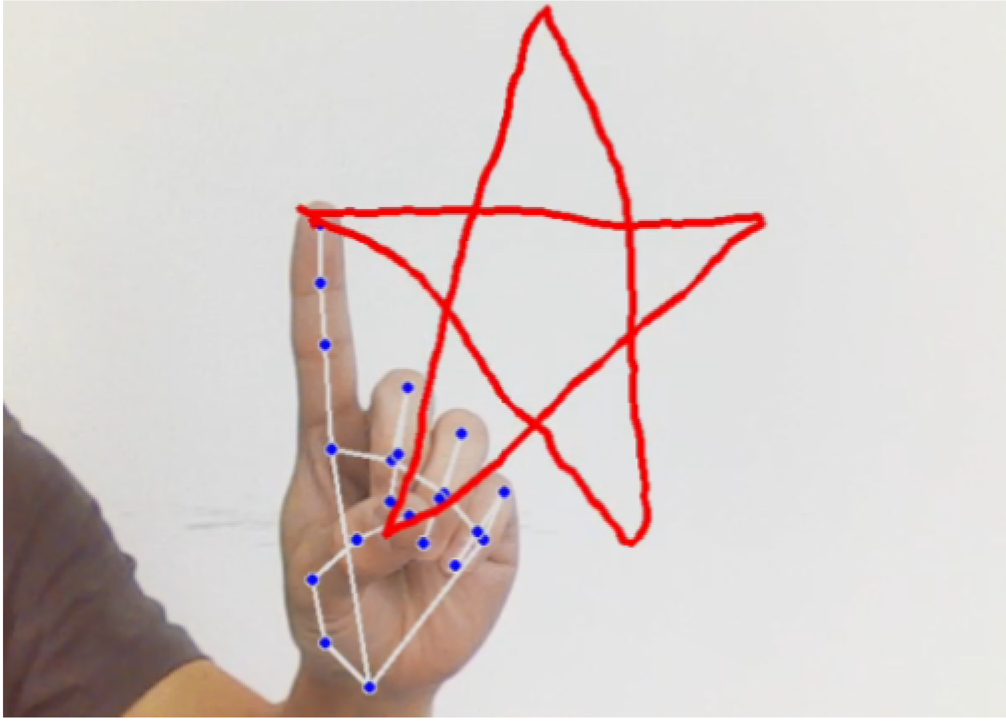
After running the program, as shown in the image below, place your palm flat on the camera screen, fingers spread out, palm facing the camera, similar to the number 5 gesture. The image will depict the joints of the entire palm. Adjust the position of your palm, aiming for the upper center of the screen.



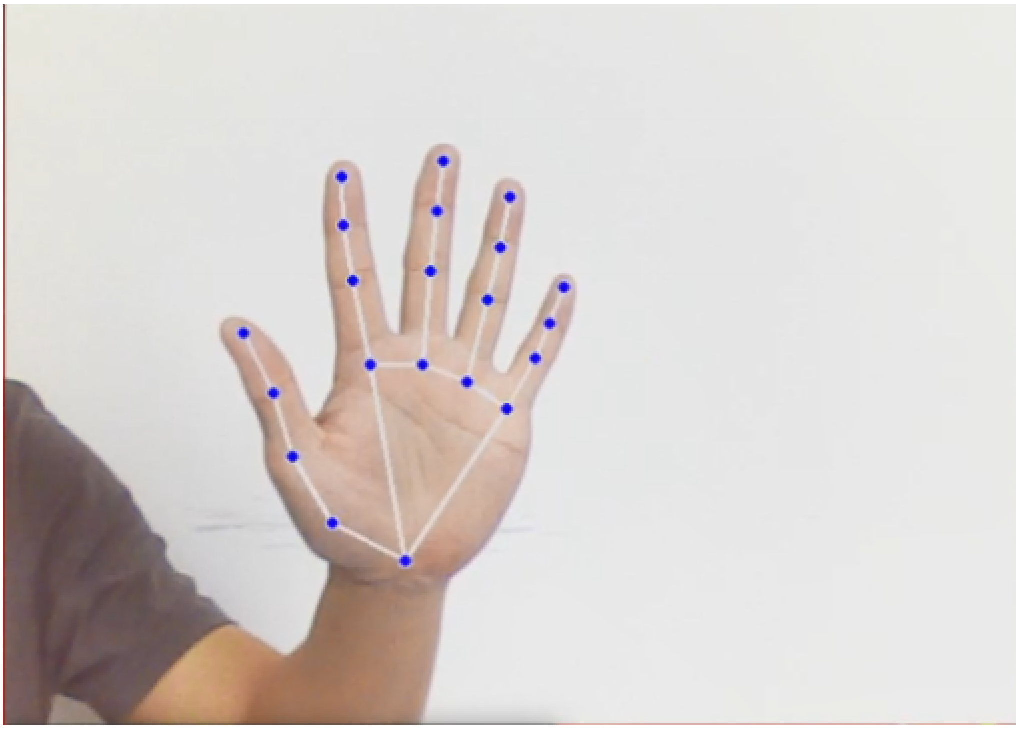
Keep your index finger still and retract the other fingers, similar to the number 1 gesture.

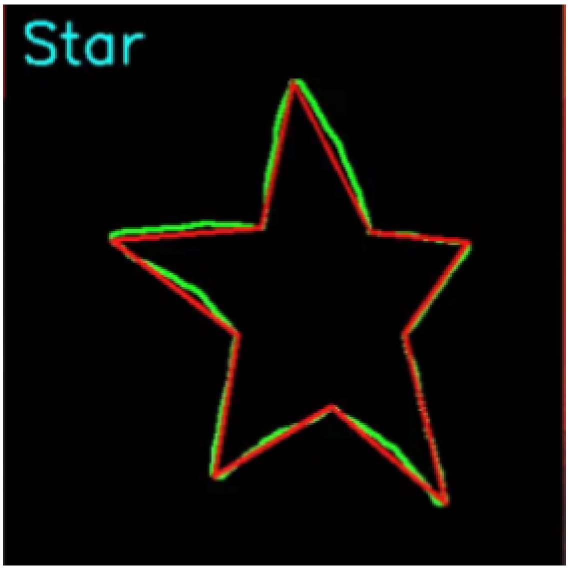


While maintaining the same position as in gesture 1, move your finger. A red line will appear on the screen, drawing the path of your index finger.



Once the graphic is drawn, open all A finger gesture similar to the number 5 will generate a drawn shape below.





Note: The drawn shape must be closed; otherwise, some content may be lost.

Currently, four trajectory shapes can be recognized: triangle, rectangle, circle, and five-pointed star.

3. Core Code Analysis

Program Code Path:

- Raspberry Pi 5 and Jetson Nano Board

  The program code is running in Docker. The path in Docker is `/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/12_FingerTrajectory.py`

- Orin board

  The program code path is `/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/12_FingerTrajectory.py`

- RDK X5

  The program code path is `/home/sunrise/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_mediapipe/yahboomcar_mediapipe/12_FingerTrajectory.py`

Import the library files used,

```
import os
import enum
import cv2
import time,math
import numpy as np
import mediapipe as mp
import rclpy
from rclpy.node import Node
import queue
from sensor_msgs.msg import Image
import gc
from cv_bridge import CvBridge
```

Initialize data and define publishers and subscribers,

```
def __init__(self):
    super().__init__('finger_trajectory')
    self.drawing = mp.solutions.drawing_utils
    self.timer = time.time()

    self.hand_detector = mp.solutions.hands.Hands(
        static_image_mode=False,
        max_num_hands=1,
        min_tracking_confidence=0.05,
        min_detection_confidence=0.6
    )
    self.bridge = CvBridge()

    #Define subscribers for the color image topic
    camera_type = os.getenv('CAMERA_TYPE', 'usb')
    topic_name = '/ascamera_hp60c/camera_publisher/rgb0/image' if camera_type ==
'nuwa' else '/usb_cam/image_raw'

    self.subscription = self.create_subscription(
        Image,
        topic_name,
        self.image_callback,
        10)
```

Color image callback function,

```
def image_callback(self,msg):
    frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
    rgb_image = cv2.flip(frame, 1) # Flip Horizontal
    result_image = np.copy(rgb_image)
    if self.timer <= time.time() and self.state == State.RUNNING:
        self.state = State.NULL
    try:
        #Call the process function to detect the palm
        results = self.hand_detector.process(rgb_image) if self.state !=
State.RUNNING else None
        #Determine whether the palm is detected. If results is not 0, it means
that the palm is detected.
        if results is not None and results.multi_hand_landmarks:
            gesture = "none"
            index_finger_tip = [0, 0]
            # Note the current time for timeout processing
            self.no_finger_timestamp = time.time()
            #Traversing the detection results
            for hand_landmarks in results.multi_hand_landmarks:
                self.drawing.draw_landmarks(
                    result_image,
                    hand_landmarks,
                    mp.solutions.hands.HAND_CONNECTIONS)
                landmarks = get_hand_landmarks(rgb_image,
hand_landmarks.landmark)
                #Call hand_angle to calculate the bending angle of each finger
                angle_list = (hand_angle(landmarks))
                #Call the h_gesture function to determine the gesture made by the
finger
                gesture = (h_gesture(angle_list))
                index_finger_tip = landmarks[8].tolist()

            if self.state == State.NULL:
                 # Detects that only the index finger is extended and the other
fingers are clenched into a fist
                if gesture == "one":
                    self.start_count += 1
                    if self.start_count > 20:
                        self.state = State.TRACKING
                        self.points = []
                else:
                    self.start_count = 0

            elif self.state == State.TRACKING:
                # Stretch out your five fingers to end drawing
                if gesture == "five":
                    self.state = State.NULL

                    # Generate black and white trajectory map
                    track_img = get_track_img(self.points)

                    contours = cv2.findContours(track_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)[-2]

                    contour = get_area_max_contour(contours, 300)

                    contour = contour[0]
                    # Identify the drawn figure according to the trajectory
diagram
                    # cv2.fillPoly draws and fills a polygon on an image
                    track_img = cv2.fillPoly(track_img, [contour,], (255, 255,
255))

                    for _ in range(3):
                        # Corrosion function
                        track_img = cv2.erode(track_img,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))

                        # Dilation function
                        track_img = cv2.dilate(track_img,
cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5)))

                    contours = cv2.findContours(track_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)[-2]

                    contour = get_area_max_contour(contours, 300)

                    contour = contour[0]
                    h, w = track_img.shape[:2]

                    track_img = np.full([h, w, 3], 0, dtype=np.uint8)
                    track_img = cv2.drawContours(track_img, [contour, ], -1, (0,
255, 0), 2)

                    # Perform polygon fitting on image contour points
                    approx = cv2.approxPolyDP(contour, 0.026 *
cv2.arcLength(contour, True), True)
```

```
                    track_img = cv2.drawContours(track_img, [approx, ], -1, (0,
0, 255), 2)

                    graph_name = 'unknown'

                    # Determine the shape based on the number of vertices of the
outline
                    if len(approx) == 3:
                        graph_name = 'Triangle'
                    if len(approx) == 4 or len(approx) == 5:
                        graph_name = 'Square'
                    if 5 < len(approx) < 10:
                        graph_name = 'Circle'
                    if len(approx) == 10:
                        graph_name = 'Star'
                    cv2.putText(track_img, graph_name, (10,
40),cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
                    cv2.imshow('track', track_img)

                else:
                    if len(self.points) > 0:
                        if distance(self.points[-1], index_finger_tip) > 5:
                            self.points.append(index_finger_tip)
                    else:
                        self.points.append(index_finger_tip)

                draw_points(result_image, self.points)
            else:
                pass
        else:
            if self.state == State.TRACKING:
                if time.time() - self.no_finger_timestamp > 2:
                    self.state = State.NULL
                    self.points = []

    except BaseException as e:
        print(e)

    #result_image = cv2.cvtColor(result_image, cv2.COLOR_RGB2BGR)
    cv2.imshow('image', result_image)
    key = cv2.waitKey(1)
     # Press the spacebar to clear the recorded tracks
    if key == ord(' '):
        self.points = []
    if time.time() > self.gc_stamp:
        self.gc_stamp = time.time() + 1
        gc.collect()
```

`hand_angle` function, calculates the bending angle of each finger

```
def hand_angle(landmarks):
    angle_list = []
    # thumb 大拇指
    angle_ = vector_2d_angle(landmarks[3] - landmarks[4], landmarks[0] -
landmarks[2])
    angle_list.append(angle_)
    # index 食指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[6], landmarks[7] -
landmarks[8])
    angle_list.append(angle_)
    # middle 中指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[10], landmarks[11] -
landmarks[12])
    angle_list.append(angle_)
    # ring 无名指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[14], landmarks[15] -
landmarks[16])
    angle_list.append(angle_)
    # pink 小拇指
    angle_ = vector_2d_angle(landmarks[0] - landmarks[18], landmarks[19] -
landmarks[20])
    angle_list.append(angle_)
    angle_list = [abs(a) for a in angle_list]
    return angle_list
```

`h_gesture` function, which determines the gesture of the finger through two-dimensional features

```
def h_gesture(angle_list):
    thr_angle, thr_angle_thumb, thr_angle_s = 65.0, 53.0, 49.0
    if (angle_list[0] < thr_angle_s) and (angle_list[1] < thr_angle_s) and
(angle_list[2] < thr_angle_s) and (
            angle_list[3] < thr_angle_s) and (angle_list[4] < thr_angle_s):
        gesture_str = "five"
    elif (angle_list[0] > 5) and (angle_list[1] < thr_angle_s) and
(angle_list[2] > thr_angle) and (
            angle_list[3] > thr_angle) and (angle_list[4] > thr_angle):
        gesture_str = "one"
    else:
        gesture_str = "none"
    return gesture_str
```

`get_track_img` function, generates the track map,

```
def get_track_img(points):
    points = np.array(points).astype(dtype=np.int32)
    x_min, y_min = np.min(points, axis=0).tolist()
    x_max, y_max = np.max(points, axis=0).tolist()
    track_img = np.full([y_max - y_min + 100, x_max - x_min + 100, 1], 0,
dtype=np.uint8)
    points = points - [x_min, y_min]
    points = points + [50, 50]
    draw_points(track_img, points, 1, (255, 255, 255))
    return track_img
```