Mediapipe gesture following

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.

1. Program Functionality

> After the program starts, it automatically detects hand gestures in the image. The robot will lock onto the detected hand gesture location (the base of the index finger) and follow it at a distance of 0.4 meters, keeping it in the center of the image. Tracking stops when a "0" (fist) gesture is detected, but tracking continues for other gestures. Pressing `Ctrl+c` in the terminal exits the program.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_depth/yahboomcar_depth/MediaPi
pe/gestureFollow.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_depth/yahboomcar_depth/media_c
ommon.py
```

- gestureFollow.py

  This function primarily performs gesture detection and following. Based on the detected coordinates of gesture key points and the depth distance between the palm and the robot, it calculates the desired turning angle and forward speed, and then sends the control angle and motor speed data to the robot.

- media_common.py
  - handDetector class:
    - Real-time hand keypoint detection and tracking
    - Gesture recognition (supports multiple gestures, including 0-8)
    - Provides finger state detection and angle calculation

3. Program Startup

3.1. Startup Commands

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).

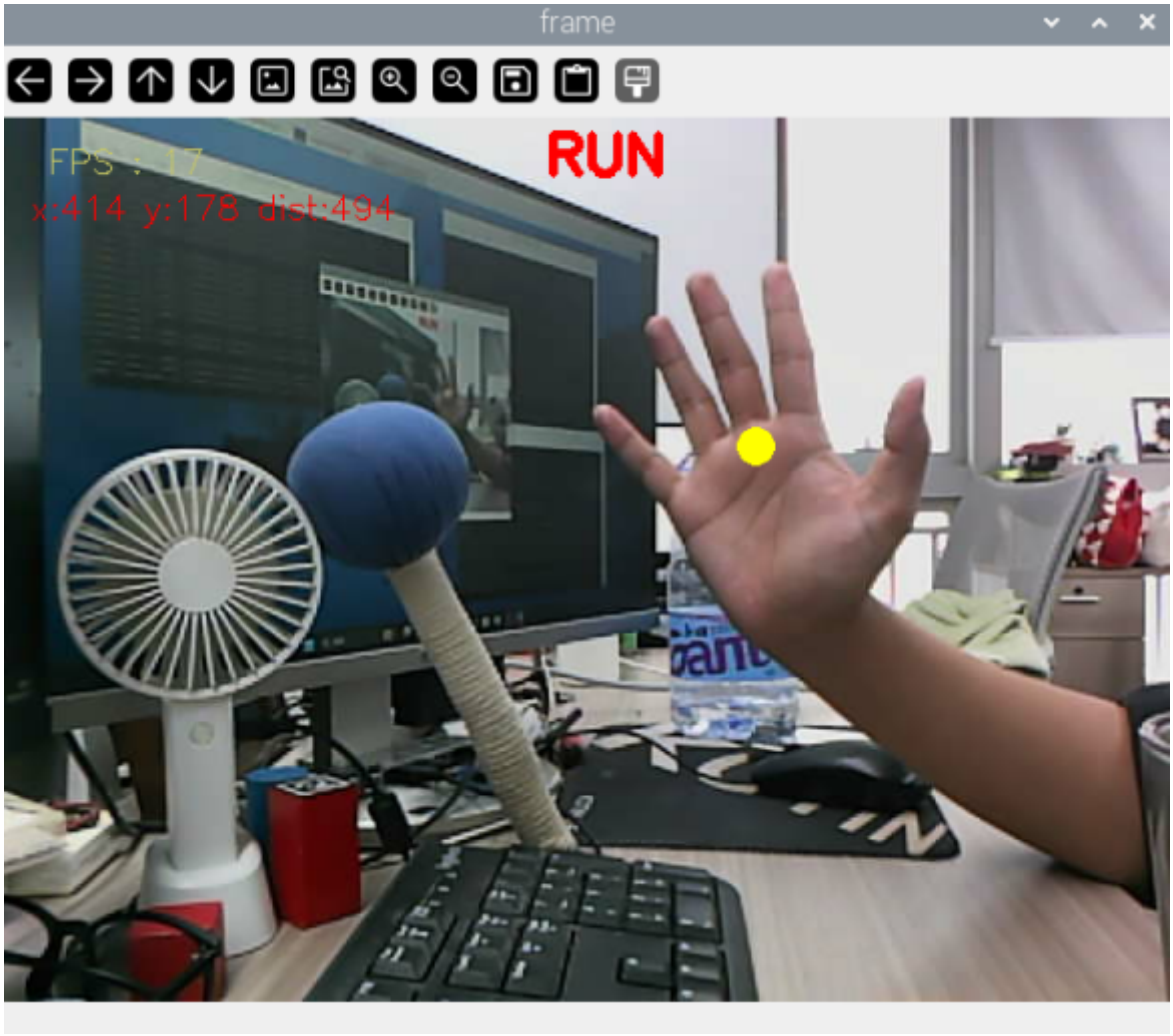
All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

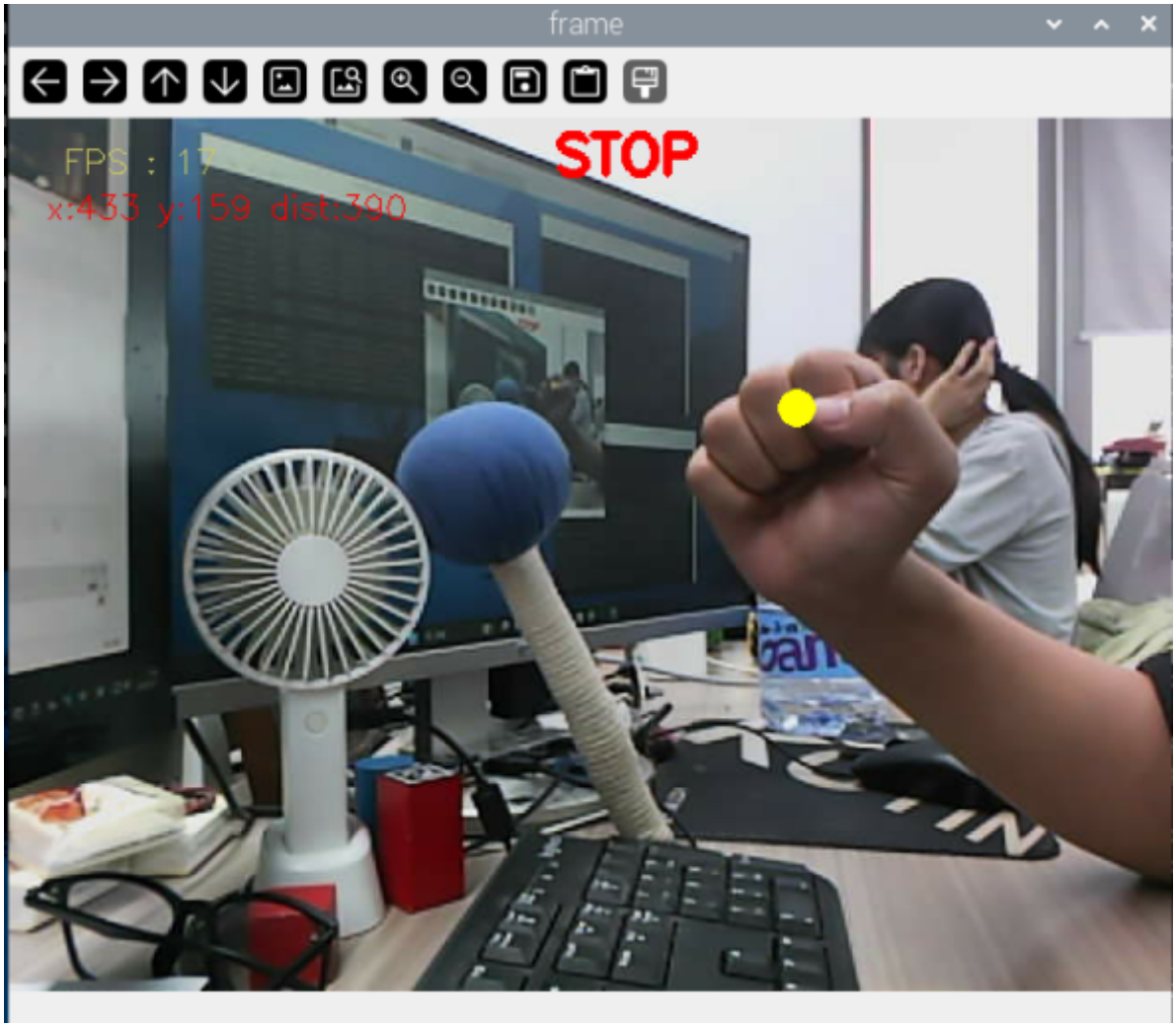
Terminal input:

```
ros2 launch ascamera hp60c.launch.py
ros2 launch yahboomcar_bringup yahboomcar_bringup_M1_launch.py
ros2 run yahboomcar_depth depth_gestureFollow
```

When a gesture is detected, the gesture location is automatically marked and its center coordinates are displayed, and the robot begins tracking. **(Note: This tracking effect is closely related to palm depth recognition. If the effect is not good, increase the tracking distance or raise the camera up.)**

After starting the program, the following screen will appear.

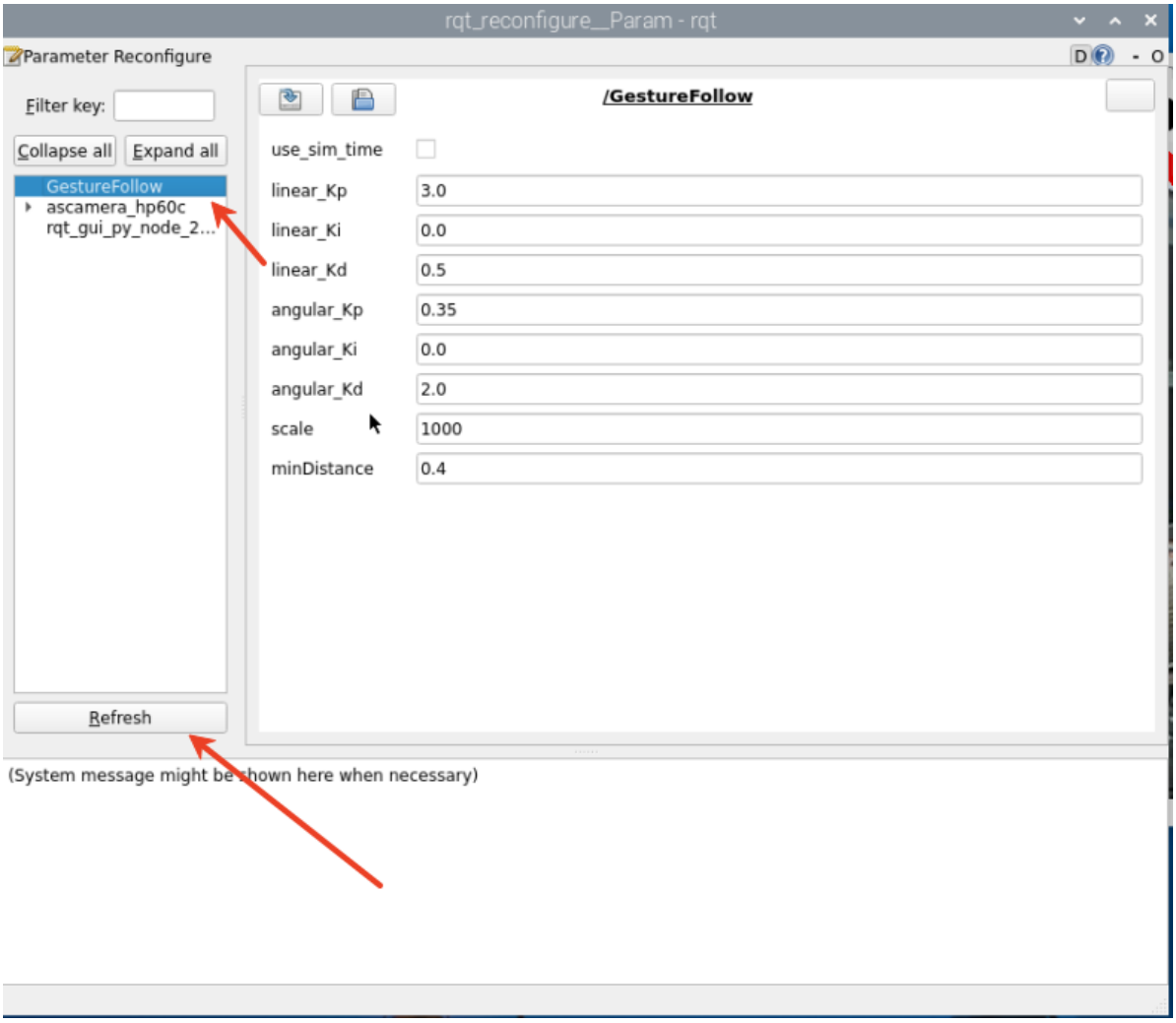




3.2 Dynamic Parameter Adjustment

In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank area in the GUI to write the parameter values. Note that the values will only take effect for the current startup. To permanently apply them, you need to modify the parameters in the source code.

Parameter Explanation:

[linear_Kp], [linear_Ki], [linear_Kd]: PID control of linear velocity during the car following process.

[angular_Kp], [angular_Ki], [angular_Kd]: PID control of angular velocity during the car following process.

[minDistance]: Following distance, which is maintained at this distance.

[scale]: PID scaling.

4. Core Code

4.1. gestureFollow.py

This program has the following main functions:

- Initializes the gesture detector
- Subscribes to a depth topic and obtains an image
- Detects gesture key points in the image
- Calculates the gesture center coordinates and publishes them
- Uses the PID algorithm to calculate the servo angle and forward speed and issues control commands

Some of the core code is as follows.

```
#初始化手掌检测对象 Initialize the palm detection object
self.hand_detector = handDetector(detectorCon=0.8)

#距离计算 #Distance calculation
points = [
        (int(self.cy), int(self.cx)),         # Center Point
        (int(self.cy +1), int(self.cx +1)),  # Lower right offset point
        (int(self.cy - 1), int(self.cx - 1))   # Upper left offset point
    ]
valid_depths = []
for y, x in points:
    depth_val = depth_image_info[y][x]
    if depth_val != 0:
        valid_depths.append(depth_val)
if valid_depths:
    dist = int(sum(valid_depths) / len(valid_depths))
else:
    dist = 0

#Determine whether to move forward or stop
if self.hand_detector.get_gesture() == "Zero":
    cv.putText(result_image, "STOP", (300, 30), cv.FONT_HERSHEY_SIMPLEX, 1, (0,
0, 255), 4)
    self.pub_cmdVel.publish(Twist())
elif dist > 0.2:
    cv.putText(result_image, "RUN", (300, 30), cv.FONT_HERSHEY_SIMPLEX, 1, (0,
0, 255), 4)
    self.execute(self.x,dist)

# 根据x值、y值，使用PID算法，计算运动速度，转弯角度
# Calculate the movement speed and turning angle using the PID algorithm based on
the x and y values
def execute(self, rgb_img, action):
    #PID计算偏差 PID calculation deviation
    linear_x = self.linear_pid.compute(dist, self.minDist)
    angular_z = self.angular_pid.compute(point_x,320)
    # 小车停车区间，速度为0 The car is in the parking area and the speed is 0
    if abs(dist - self.minDist) < 30: linear_x = 0
    if abs(point_x - 320.0) < 30: angular_z = 0
    twist = Twist()
    ...
    # 将计算后的速度信息发布 Publish the calculated speed information
    self.pub_cmdVel.publish(twist)
```