Mediapipe Posture 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 key points of human posture in the image. The robot will lock onto the detected center of the person (the average of all key points) and follow the person at a distance of 1.2 meters, keeping it in the center of the image. Pressing `Ctrl+c` in the terminal exits the program.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_depth/yahboomcar_depth/MediaPipe/poseFollow.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/common/media_common.py
```

- poseFollow.py
  This mainly performs human pose detection and following. Based on the detected pose keypoint coordinates and the depth distance between the human pose center and the car, it calculates the car's desired turning angle and forward speed, and sends control angle and motor speed data to the car.
- media_common.py

  This is a gesture and posture recognition system based on MediaPipe.

  - PoseDetector class:
    - Human pose estimation and keypoint detection
    - Returns 3D coordinates of 33 body keypoints
    - Calculates the bounding box area of the detected region

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.

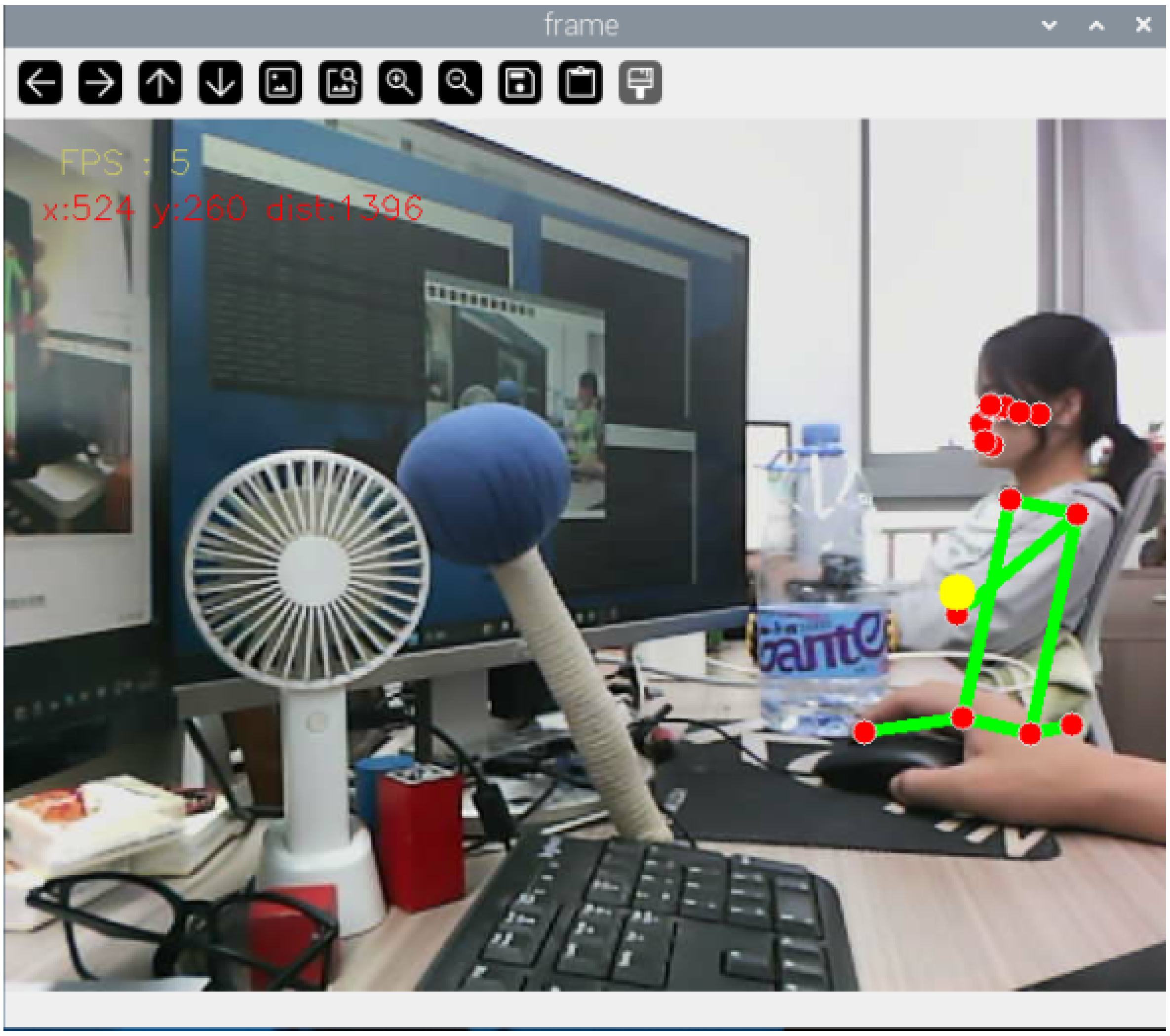
Enter the Docker container (see [Docker course] --- [4. Docker Startup Script] for steps).

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

In the terminal, enter:

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

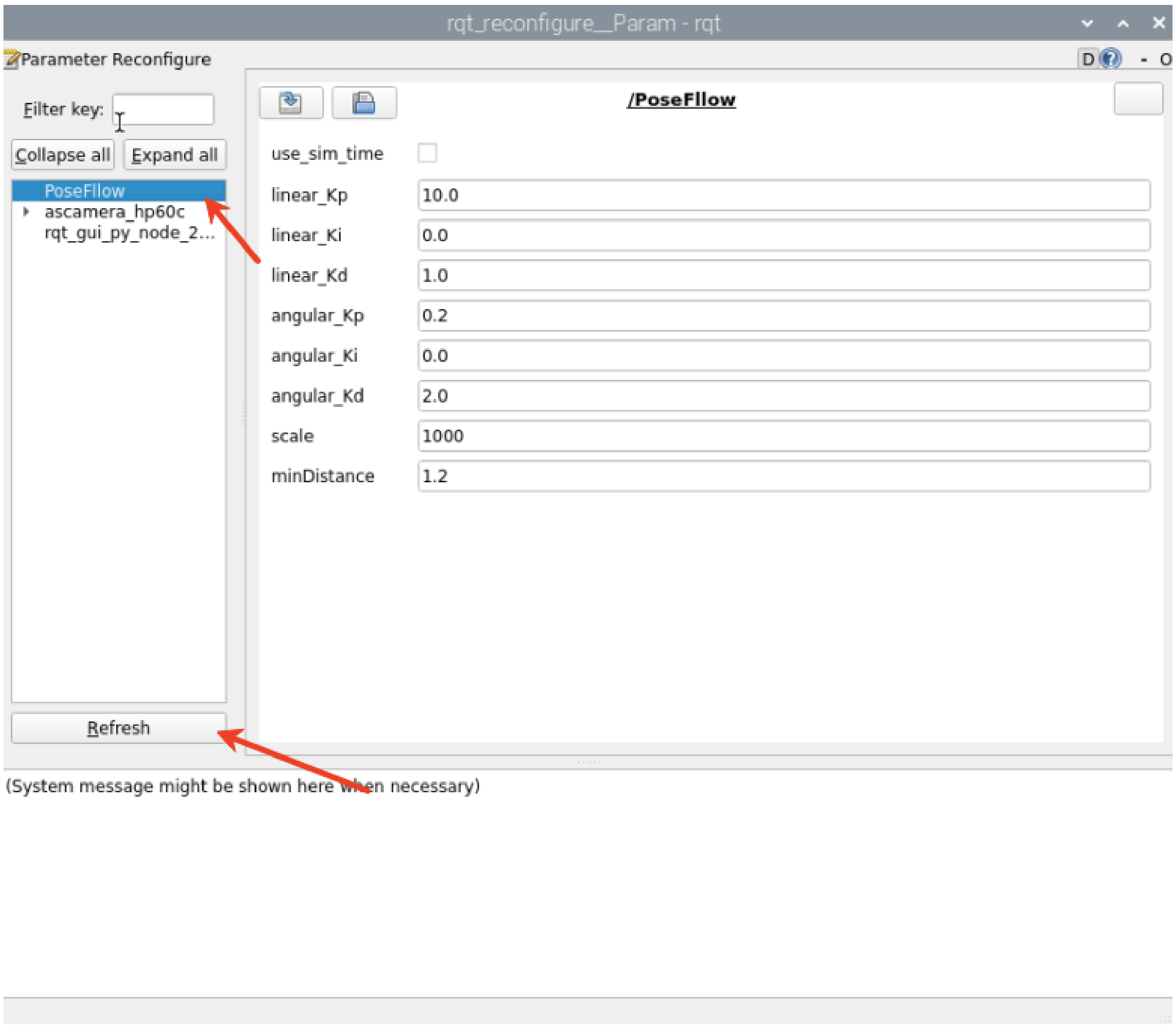
After starting the program, the following screen will appear. When a person is detected, key points are automatically marked and the center position of all average points is calculated. The robot then begins following.



3.2 Dynamic Parameter Adjustment

In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



use_sim_time: ☐
linear_Kp: 10.0
linear_Ki: 0.0
linear_Kd: 1.0
angular_Kp: 0.2
angular_Ki: 0.0
angular_Kd: 2.0
scale: 1000
minDistance: 1.2

(System message might be shown here when necessary)

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

Parameter Analysis:

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

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

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

[scale]: PID scaling.

4. Core Code

4.1. depth_poseFollow.py

This program has the following main functions:

- Initializes the pose detector
- Subscribes to the depth topic and obtains images
- Detects pose keypoints in the image
- Calculates the pose center coordinates and publishes them
- Calculates the servo angle and forward speed using the PID algorithm and issues control commands

Part of the core code is as follows,

```
#Initialize the posture detection object
self.pose_detector = PoseDetector()

#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
#Follow the running program
if dist > 0.2:
   self.execute(center_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)
```