Meidpipe human posture tracking

1. Program Functionality

> After the program starts, it automatically detects key points of human poses in the image. The servo gimbal will lock onto the detected center position of the human body (the average of all key points), keeping it at the center of the image. Press `Ctrl+c` in the terminal to exit the program.

2. Program Code Reference Path

```
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/poseTracker.py
~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_astra/yahboomcar_astra/common/media_common.py
```

- poseTracker.py
  Mainly performs human pose detection and tracking. It calculates the center coordinates based on the detected key points, calculates the desired servo rotation angle, and sends the servo angle control data to the robot.
- media_common.py

  This is a gesture and posture recognition system based on MediaPipe.
- PoseDetector class:
- Human pose estimation and keypoint detection
- Returns the 3D coordinates of 33 body keypoints
- Calculates the bounding box area of the detection region

3. Program Startup

3.1. Startup Commands

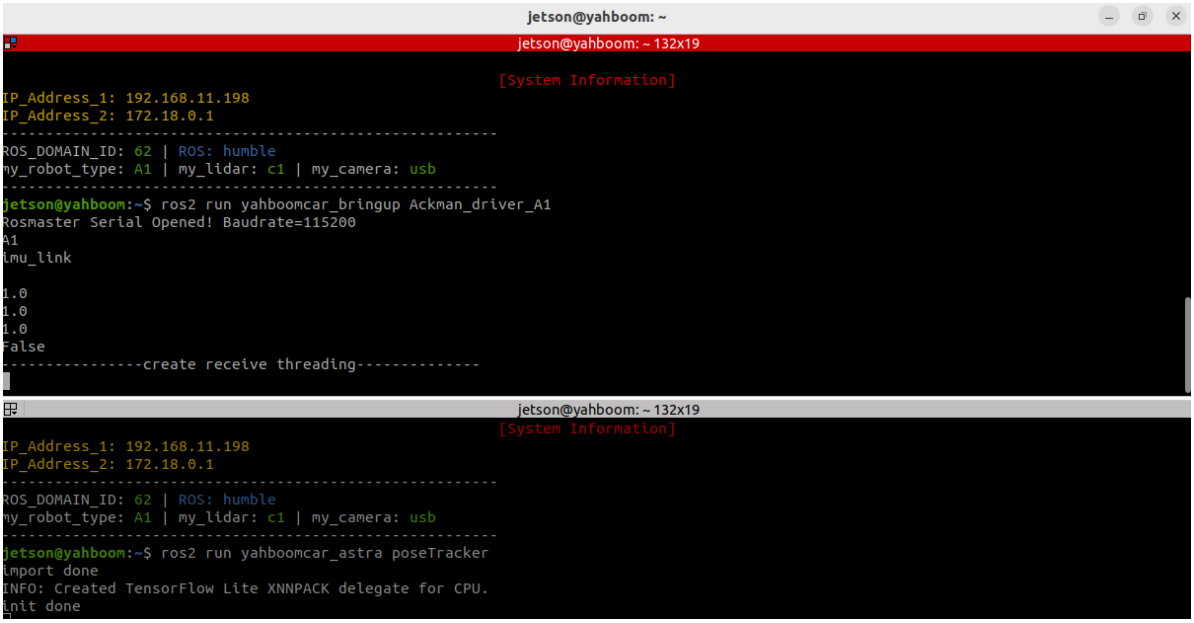
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

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

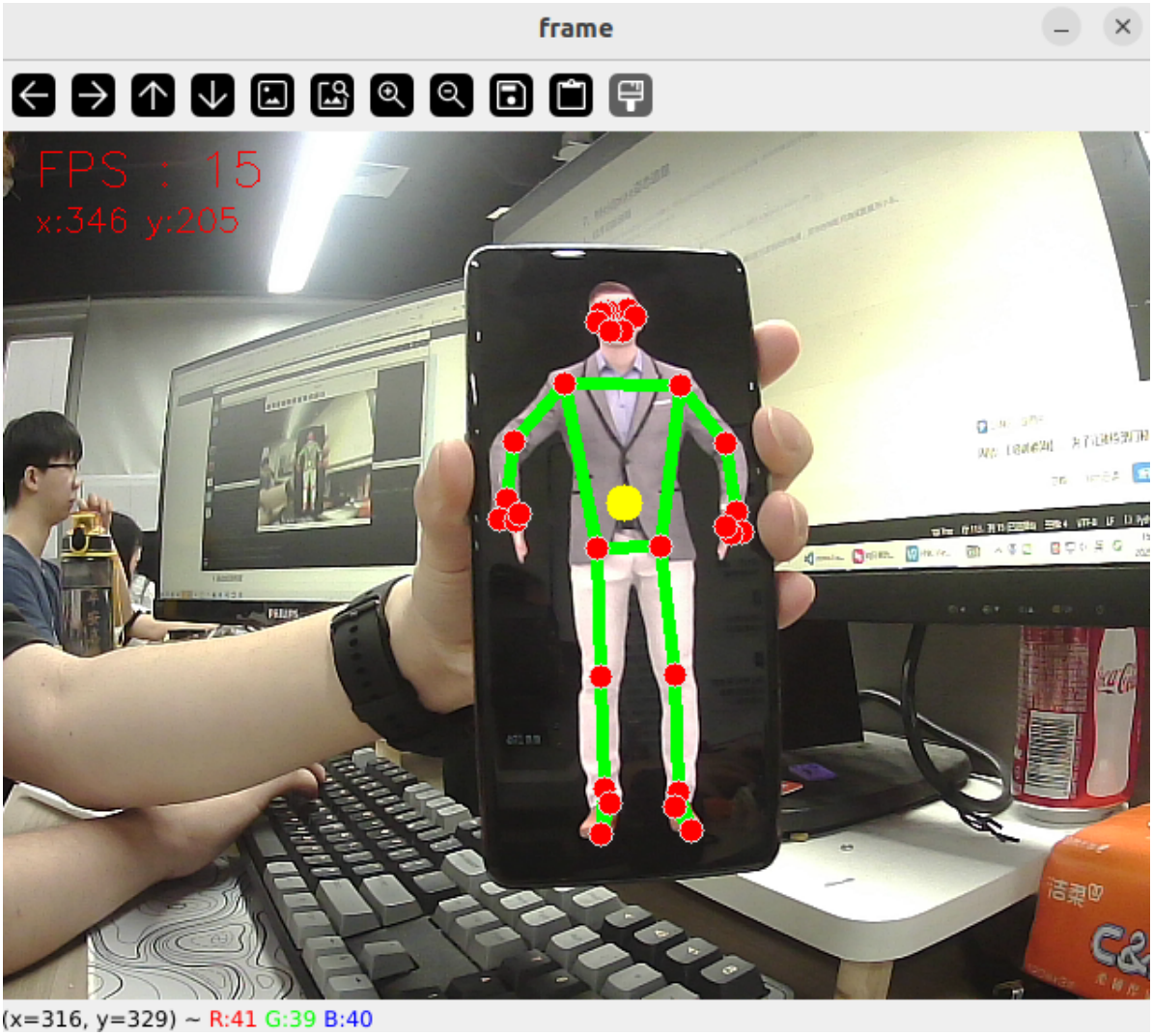
All the following commands must be executed within the same Docker container **(see [Docker course] --- [3. Docker Submission and Multi-Terminal Access] for steps).**

Enter the terminal:

```
# Start the car chassis
ros2 run yahboomcar_bringup Mcnamu_driver_M1
# Start the human pose tracking program
ros2 run yahboomcar_astra poseTracker
```



When a person is detected, key points are automatically marked and the center position of all average points is calculated. The servo gimbal then begins tracking.



After starting the program, the following screen will appear.

In addition, you can also enter the following command to print information about the target center coordinates.

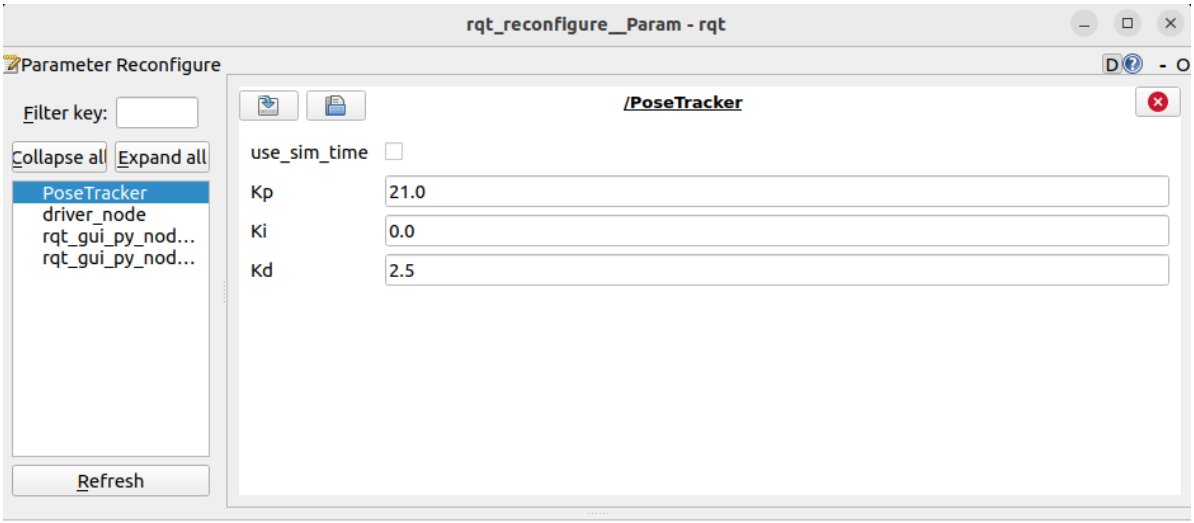
```
ros2 topic echo /Current_point
```

```
jetson@yahboom:~$ ros2 topic echo /Current_point
anglex: 42.0
angley: 220.0
distance: 0.0
---
anglex: 49.0
angley: 222.0
distance: 0.0
---
```

3.2. Dynamic Parameter Adjustment

In the terminal, enter:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



☑ After modifying the parameters, click a blank space in the GUI to enter the parameter value. Note that this will only take effect during the current boot. To permanently effect the value, you need to modify the parameters in the source code.

As shown in the figure above,

- poseTracker is primarily responsible for servo gimbal movement. PID-related parameters can be adjusted to achieve optimal gimbal motion.

🛠 Parameter Analysis:

[Kp], [Ki], [Kd]: PID control of the servo gimbal speed during tracking.

4. Core Code

4.1. poseTracker.py

This program has the following main functions:

- Initialize the pose detector
- Open the camera and capture an image
- Detect human pose keypoints in the image
- Calculate the average center coordinates of all keypoints and publish them
- Use the PID algorithm to calculate the servo angle and issue control commands

Some core code is as follows:

```
# Create a publisher to publish the center coordinates of the tracked object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
# Define the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
...
# Initialize the pose detector
self.pose_detector = PoseDetector()
...
# Detect pose keypoints and calculate the center position
frame, pose_points, _= self.pose_detector.pubPosePoint(frame)
if pose_points:
    sum_x = sum(point[1] for point in pose_points)
    sum_y = sum(point[2] for point in pose_points)
    center_x = sum_x / len(pose_points)
```

```
    center_y = sum_y / len(pose_points)
    threading.Thread(target=self.execute, args=(center_x, center_y)).start()
...
# According to the x value and y value, use the PID algorithm to calculate the
servo angle
def execute(self, point_x, point_y):
    position = Position()
    position.anglex = point_x * 1.0
    position.angley = point_y * 1.0
    # Publish center coordinates message
    self.pub_position.publish(position)
    ...
    # Limit PID polarity and maximum value
    [x_Pid, y_Pid] = self.PID_controller.update([point_x - 320, point_y - 240])
    x_Pid = x_Pid * (abs(x_Pid) <= self.Kp/2.5)
    y_Pid = y_Pid * (abs(y_Pid) <= self.Kp/2.5)
    ...
    # Publish the calculated servo angle
    self.pub_Servo.publish(self.servo_angle)
```