Meidpipe human posture following

1. Program Functionality

> After the program starts, it automatically detects the human pose in the image. The robot enters follow mode, and the servos and gimbal lock onto the center of the detected human pose (torso center), keeping it in the center of the image. Following stops when the human pose is no longer detected. Use the Ctrl+c shortcut key in the terminal to exit the program. The R2 key on the remote controller has a [Pause/Enable] function for this function.
>
> 📝 The following object example uses a real person. If you use a smaller follow object, you will need to adjust the corresponding parameters to achieve your desired effect!

2. Program Code Reference Path

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

- poseFollow.py
  This mainly performs human pose detection. Based on the detected pose center coordinates, it calculates the required servo rotation angle and the required vehicle angular velocity. It also calculates the required vehicle linear velocity based on the pose area size. After PID calculation, it publishes the data to the vehicle chassis.
- 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 Command

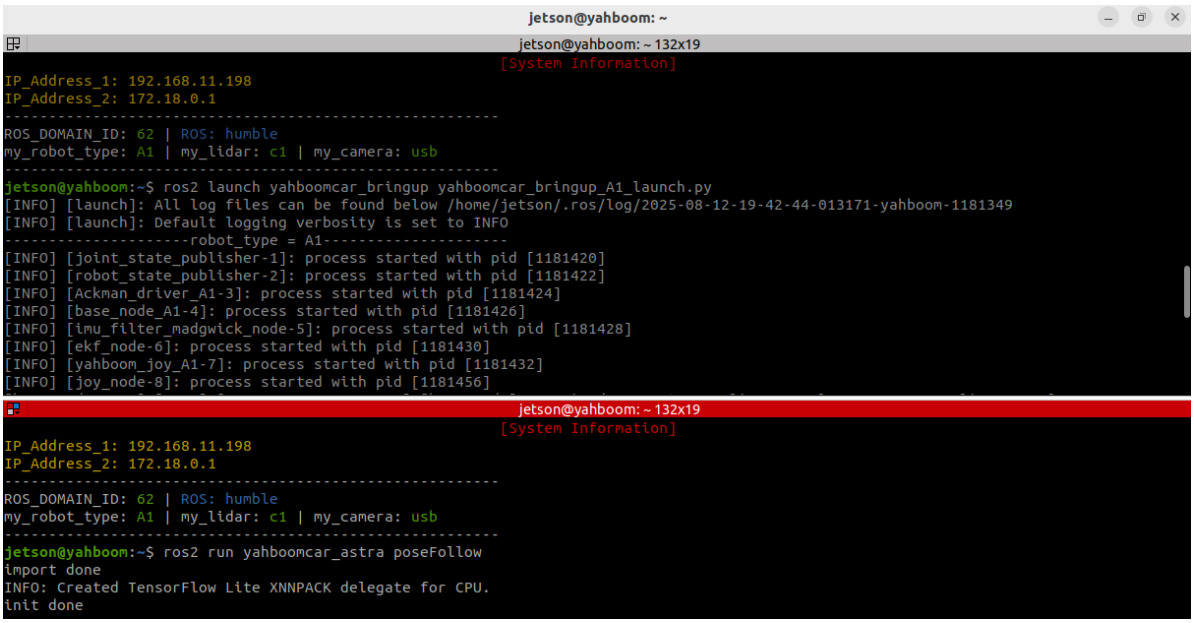
For Raspberry Pi and Jetson-Nano boards, you need to enter the Docker container first. For RDKX5 and Orin controllers, this is not necessary.

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).**

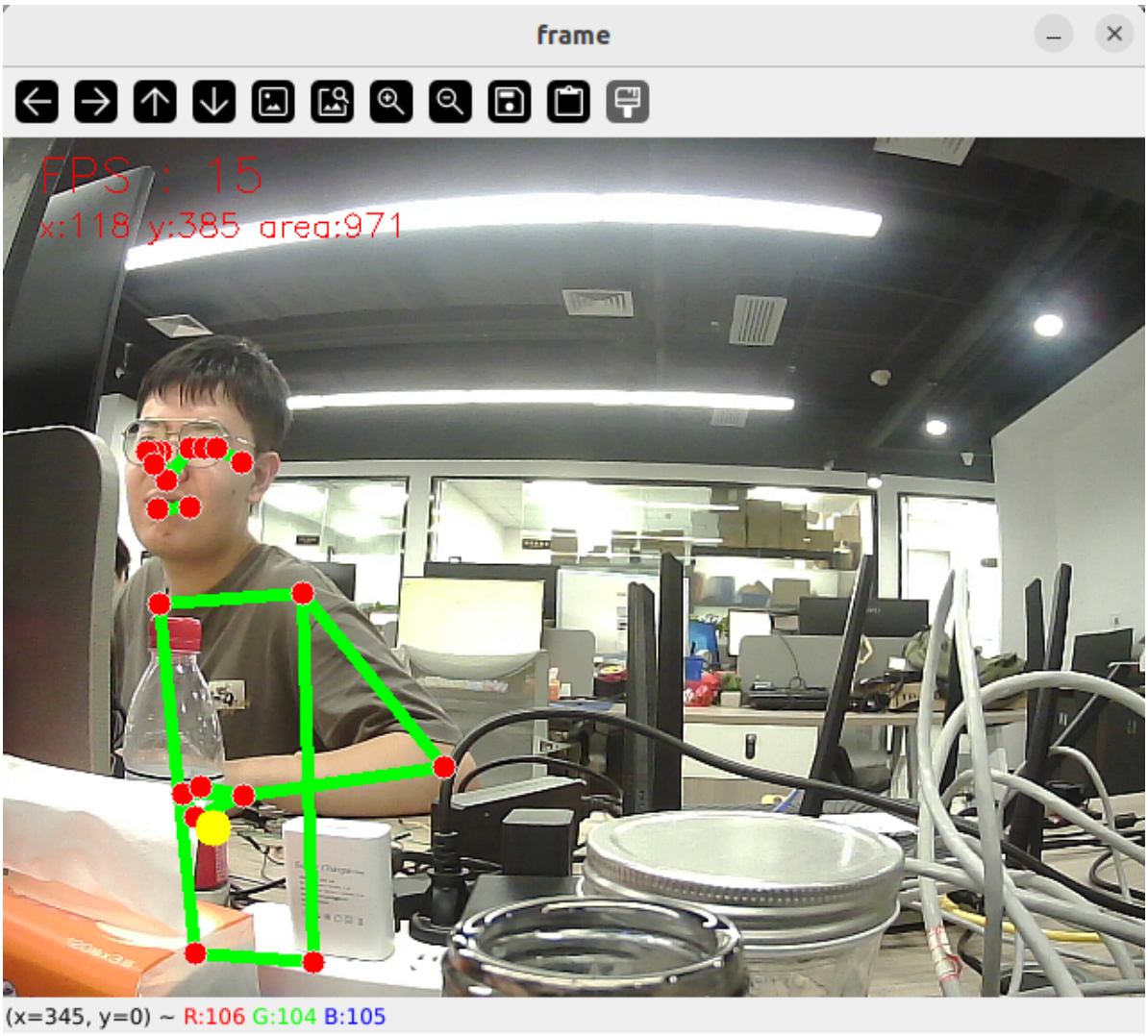
Enter the terminal.

```
# Start the car chassis and joystick nodes
ros2 launch yahboomcar_bringup yahboomcar_bringup_M1_launch.py
# Start the human pose tracking program
ros2 run yahboomcar_astra poseFollow
```



After starting the program, the camera image will appear and the human posture will be detected in real time.

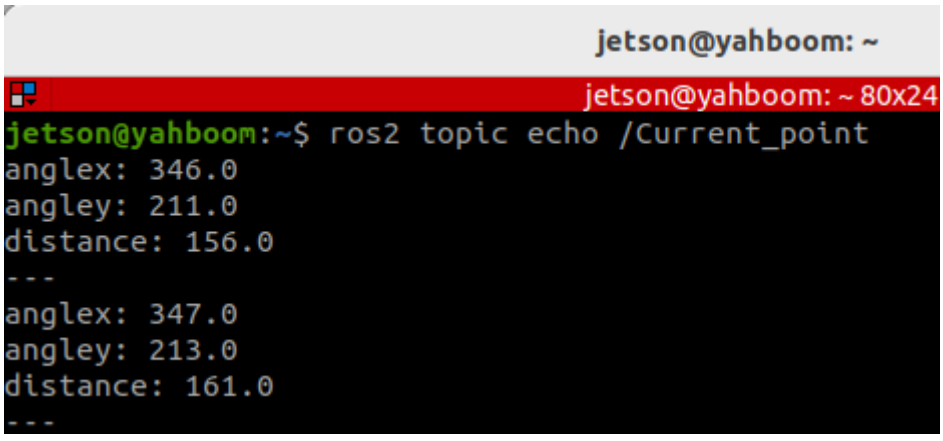
When a human posture is detected, a yellow dot will be drawn at the center of the torso, and the center coordinates and the size of the human posture area will be displayed.



Then, the robot enters follow mode. Slowly move your body, and the robot and servo gimbal will follow.

Alternatively, we can enter the following command to print the target center coordinates and area information:

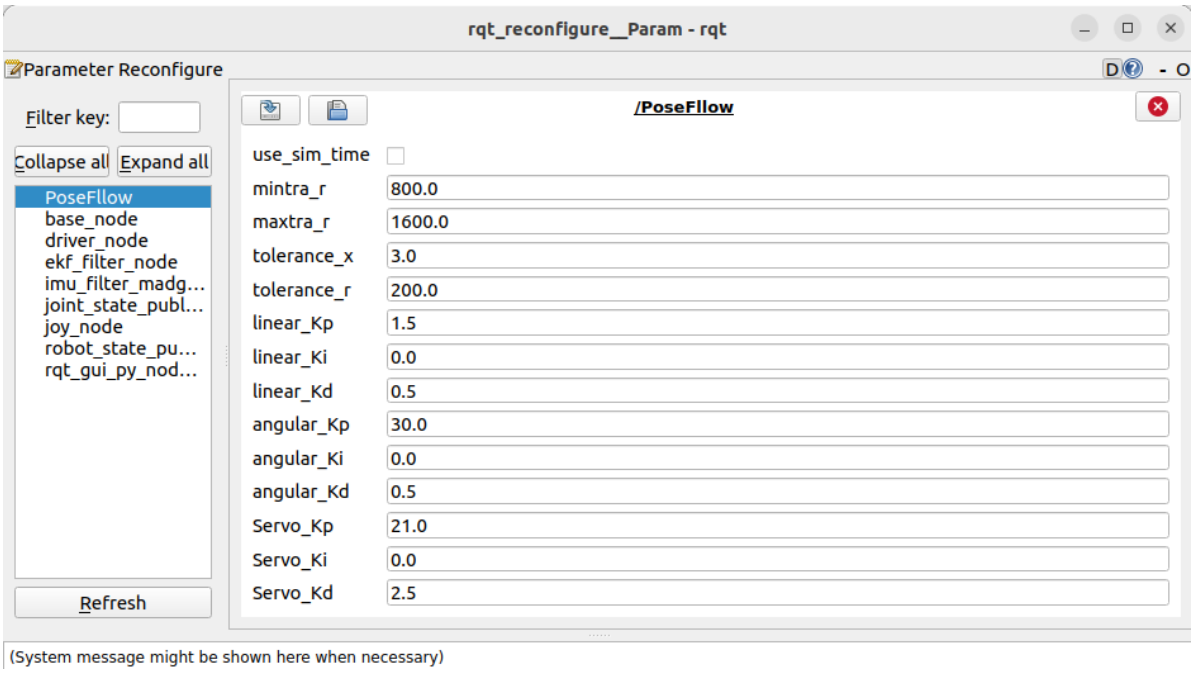
```
ros2 topic echo /Current_point
```



3.2 Dynamic Parameter Adjustment

Enter in the terminal:

```
ros2 run rqt_reconfigure rqt_reconfigure
```



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

As shown in the figure above:

- poseFollow is primarily responsible for the robot's motion. PID-related parameters can be adjusted to optimize the robot's motion.

🛠 Parameter Explanation:

[mintra_r]: Critical tracking area. When the area of the human pose region is smaller than this value, the robot advances; when it is larger than this value, the robot retreats.

[maxtra_r]: Maximum tracking area. When the area of the human pose region is larger than this value, tracking is stopped. **(To prevent large objects from affecting the judgment when tracking small objects)**

[tolerance_x], [tolerance_r]: X-axis position tolerance. When the deviation between the pose center's x coordinate and the screen center (`320`) is less than this value, alignment is considered achieved. Area tolerance: When the deviation between the detected pose region's area and the critical tracking area (`mintra_r`) is less than this value, the distance is considered appropriate.

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

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

[Servo_Kp], [Servo_Ki], [Servo_Kd]: PID control of speed during the servo gimbal following process.

4. Core Code

4.1. poseFollow.py

This program has the following main functions:

- Opens the camera and acquires images;
- Uses the mediapipe library to detect human poses;
- Calculates the pose center coordinates (torso center) and region size;
- Controls the following state based on the detected status;
- Calculates the car's movement speed and servo angle, and issues control commands.

Some core code is as follows:

```
# Define the servo data publisher
self.pub_Servo = self.create_publisher(ServoControl, 'Servo', 10)
# Define the vehicle velocity publisher
self.pub_cmdVel = self.create_publisher(Twist, '/cmd_vel', 10)
# Create a publisher to publish the center coordinates and radius of the tracked
object
self.pub_position = self.create_publisher(Position, "/Current_point", 10)
# Define the controller node data subscriber
self.sub_JoyState = self.create_subscription(Bool,"/JoyState",
self.JoyStateCallback, 1)
...
# Initialize the pose detector
self.pose_detector = PoseDetector()
...
# Pose detection
frame, pose_points, center_r = self.pose_detector.pubPosePoint(frame)
if pose_points and center_r < self.maxtra_r:
    # Calculate the coordinates of the posture center
    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)
    cv.circle(frame, (int(center_x),int(center_y)), 10, (0, 255, 255), -1)
    # Follow the posture
    self.execute(center_x, center_y, center_r)
else:
    self.execute(320, 240, self.mintra_r)  # Stop following
...

# According to the x value, y value and area size, use the PID algorithm to
calculate the car speed and servo angle
def execute(self, x, y, z=None):
    position = Position()
    position.anglex = x * 1.0
    position.angley = y * 1.0
    position.distance = z * 1.0
    self.pub_position.publish(position)
    ...
    # Calculate PID control quantity
    [linear_Pid, Servox_Pid, Servoy_Pid] = self.PID_controller.update([z -
self.mintra_r, x - 320, y - 240])
    angular_Pid = self.angular_PID_controller.update([self.PWMServo_X -
self.init_servos1])[0]
```

```
    ...
    # Limit output range
    linear_Pid = max(-0.85, min(linear_Pid, 0.85))
    angular_Pid = max(-3.0, min(angular_Pid, 3.0))
    ...
    # Limit the servo PID polarity and maximum value
    Servox_Pid = Servox_Pid * (abs(Servox_Pid) <=self.Servo_Kp/2.5)
    Servoy_Pid = Servoy_Pid * (abs(Servoy_Pid) <=self.Servo_Kp/2.5)
    ...
    # Set the car speed and servo angle
    self.twist.linear.x = -linear_Pid
    self.twist.angular.z = -angular_Pid if self.img_flip else angular_Pid
    self.PWMServo_X += Servox_Pid if not self.img_flip else -Servox_Pid
    self.PWMServo_Y += Servoy_Pid if not self.img_flip else -Servoy_Pid
    ...
    # Issue control instructions
    self.pub_Servo.publish(self.servo_angle)
    self.pub_cmdVel.publish(self.twist)
```