7、Body movement control robot

The Raspberry Pi motherboard series does not currently support this tutorial.

7.1、Introduction to gameplay

The body control robot gameplay is a function realized by combining the poseNet module of jetson-inference and the motion control of the Muto robot. poseNet is mainly responsible for detecting the joint points of the human body in the image, and drawing the joint points of the human body through points and lines, and then the system The program analyzes the coordinate parameters of each joint point to control the robot's movement.

7.2、Core content analysis

Initialize the Muto robot and set the initial angle of the camera gimbal. The default is S1=90 and S2=60. The angle of the camera gimbal can be modified according to the actual viewing angle.

```
g_bot = Muto()
g_bot.Gimbal_1_2(90, 60)
```

Import the poseNet module from jetson-inference, and import the image input and output modules from jetson_utils.

```
from jetson_inference import poseNet
from jetson_utils import videoSource, videoOutput, Log
```

Configure input parameters：

The input source defaults to USB camera device /dev/video0. The USB camera device number can be modified according to the device number found in the system.

The output source defaults to the graphical desktop.

The network network model defaults to resnet18-pose.

The overlay layer defaults to keypoints and links.

The default detection threshold is 0.15, which can be modified according to the actual recognition effect. The recommended range is 0.1-1.0.

```
parser = argparse.ArgumentParser(description="Run pose estimation DNN on a
video/image stream.",
                                 formatter_class=argparse.RawTextHelpFormatter,
```

```
                                    epilog=poseNet.Usage() + videoSource.Usage() +
videoOutput.Usage() + Log.Usage())

parser.add_argument("input", type=str, default="/dev/video0",
                    nargs='?', help="URI of the input stream")
parser.add_argument("output", type=str, default="",
                    nargs='?', help="URI of the output stream")
parser.add_argument("--network", type=str, default="resnet18-pose",
                    help="pre-trained model to load (see below for options)")
parser.add_argument("--overlay", type=str, default="links,keypoints",
                    help="pose overlay flags (e.g. --overlay=links,keypoints)\nvalid
combinations are:  'links', 'keypoints', 'boxes', 'none'")
parser.add_argument("--threshold", type=float, default=0.15,
                    help="minimum detection threshold to use")
try:
    args = parser.parse_known_args()[0]
except:
    print("")
    parser.print_help()
    sys.exit(0)

# load the pose estimation model
net = poseNet(args.network, sys.argv, args.threshold)

# create video sources & outputs
input = videoSource(args.input, argv=sys.argv)
output = videoOutput(args.output, argv=sys.argv)
```

The IDs corresponding to the joint points detected by poseNet are as follows:

```
-- ID:  0 (nose)
-- ID:  1 (left_eye)
-- ID:  2 (right_eye)
-- ID:  3 (left_ear)
-- ID:  4 (right_ear)
-- ID:  5 (left_shoulder)
-- ID:  6 (right_shoulder)
-- ID:  7 (left_elbow)
-- ID:  8 (right_elbow)
-- ID:  9 (left_wrist)
-- ID:  10 (right_wrist)
-- ID:  11 (left_hip)
-- ID:  12 (right_hip)
-- ID:  13 (left_knee)
-- ID:  14 (right_knee)
-- ID:  15 (left_ankle)
-- ID:  16 (right_ankle)
-- ID:  17 (neck)
```

Read the camera image and then transmit it to detectNet to calculate and output the detection result.

```
img = input.Capture()
poses = net.Process(img, overlay=args.overlay)
```

Traverse the poses results, extract all the joint point information of the first detected human body in the result, and then transfer all the joint point information of this human body to the parse_action function for analysis.

```
for pose in poses:
    # print(pose)
    # print(pose.Keypoints)
    # print('Links', pose.Links)
    num_keypoints = len(pose.Keypoints)
    # print("num_keypoints", num_keypoints)
    parse_action(pose.Keypoints)
    break
```

The main function of the parse_action function is to extract the position information of all joint points in the Keypoints parameter, and then control the robot based on the difference in the current posture of the human body. It should be noted here that in order to increase control stability, the human posture must include a nose and two eyes before the robot can be controlled. The g_control_count variable is mainly used to calculate the timeout stop function of the robot movement. The robot will no longer be controlled before the robot movement times out.

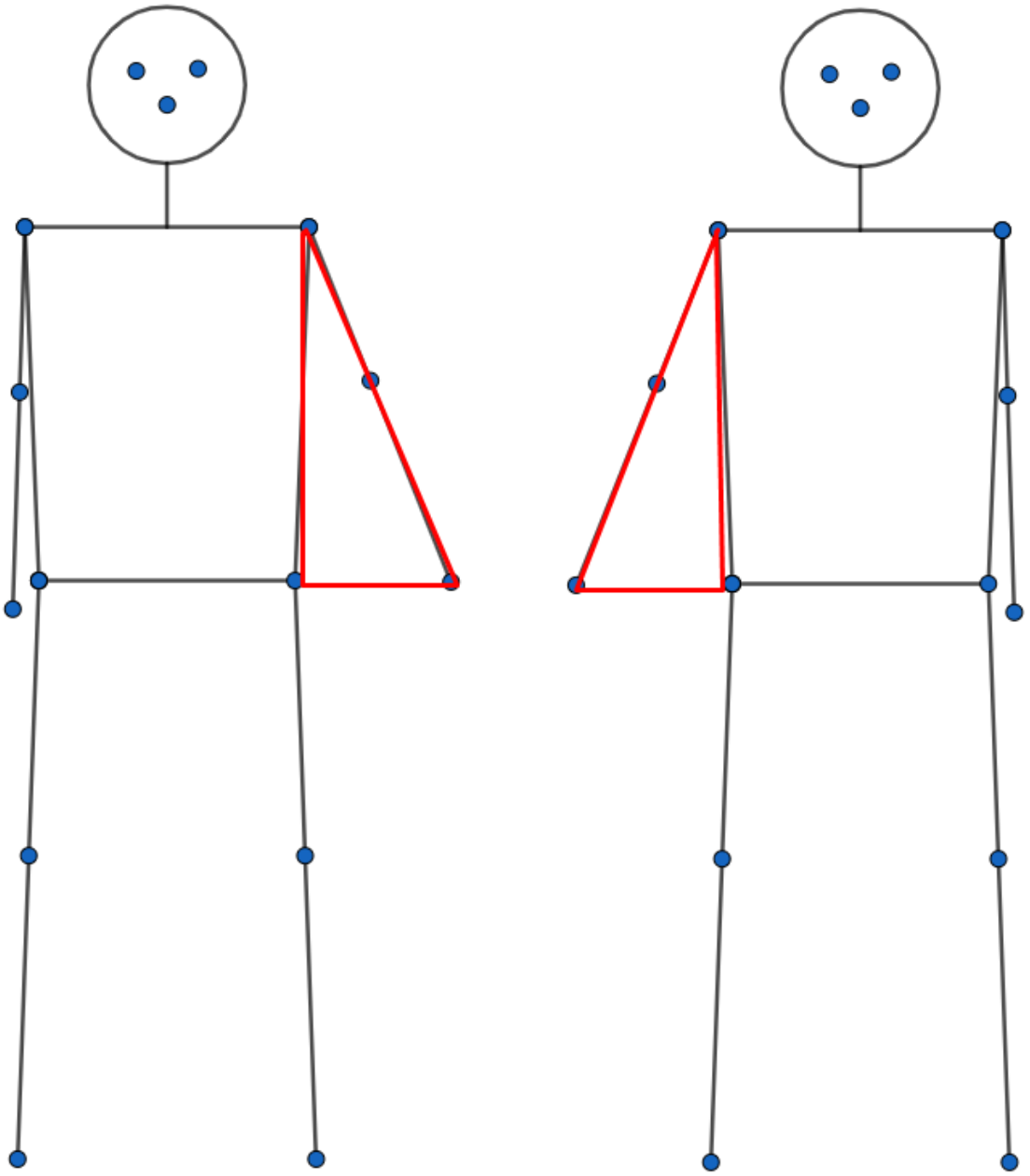
```
def parse_action(Keypoints):
    ...
    if 1 <= int(p_point.ID) <= 3:
            check = check + 1
    ...
    if g_control_count <= 0 and check == 3:
        start = control_turn_left(right_shoulder, right_hip, right_wrist)
        if start: return
        start = control_turn_right(left_shoulder, left_hip, left_wrist)
        if start: return
        start = control_left(right_shoulder, right_elbow, right_wrist)
        if start: return
        start = control_right(left_shoulder, left_elbow, left_wrist)
        if start: return
        start = control_back(left_shoulder, left_elbow, left_wrist)
        if start: return
        control_forward(right_shoulder, right_elbow, right_wrist)
```

Control the robot to rotate left. Since the robot is opposed to the human body, the human body's right hand shoulder, hip and wrist joint point position information is used to distinguish the instruction for the robot to rotate left. Reach your right hand out to the right, keeping your shoulders, hips, and wrists forming a triangle. Since each human body may have certain differences, the detection range of the three values of offset x y z can be modified according to the actual situation. offset_x represents the horizontal distance from the wrist to the hip, that is, the length of the base of the triangle, offset_y represents the deviation of the base from the horizontal plane, and offset_z

Represents the deviation of the vertical plane of the hips and shoulders, that is, the deviation of the height of the triangle from the direction of gravity.

```
def control_turn_left(shoulder, hip, wrist):
    global g_control_count, g_turn_left_count
    result = 0
    offset_x = wrist[0] - hip[0]
    offset_y = wrist[1] - hip[1]
    offset_z = shoulder[0] - hip[0]
    print("turn_left:", offset_x, offset_y, offset_z)
    if (130 < abs(offset_x) < 250 and abs(offset_y) < 100 and abs(offset_z) < 100):
        g_turn_left_count = g_turn_left_count + 1
        if g_turn_left_count > 3:
            result = 1
            g_bot.turnleft(g_step)
            g_control_count = 10
            clear_action_count()
    else:
        g_turn_left_count = 0
    return result
```

The judgment idea of controlling right rotation is the same as that of controlling left rotation function. Stretch your left hand to the left and keep your shoulders, hips and wrists forming a triangle to control the robot's right rotation.

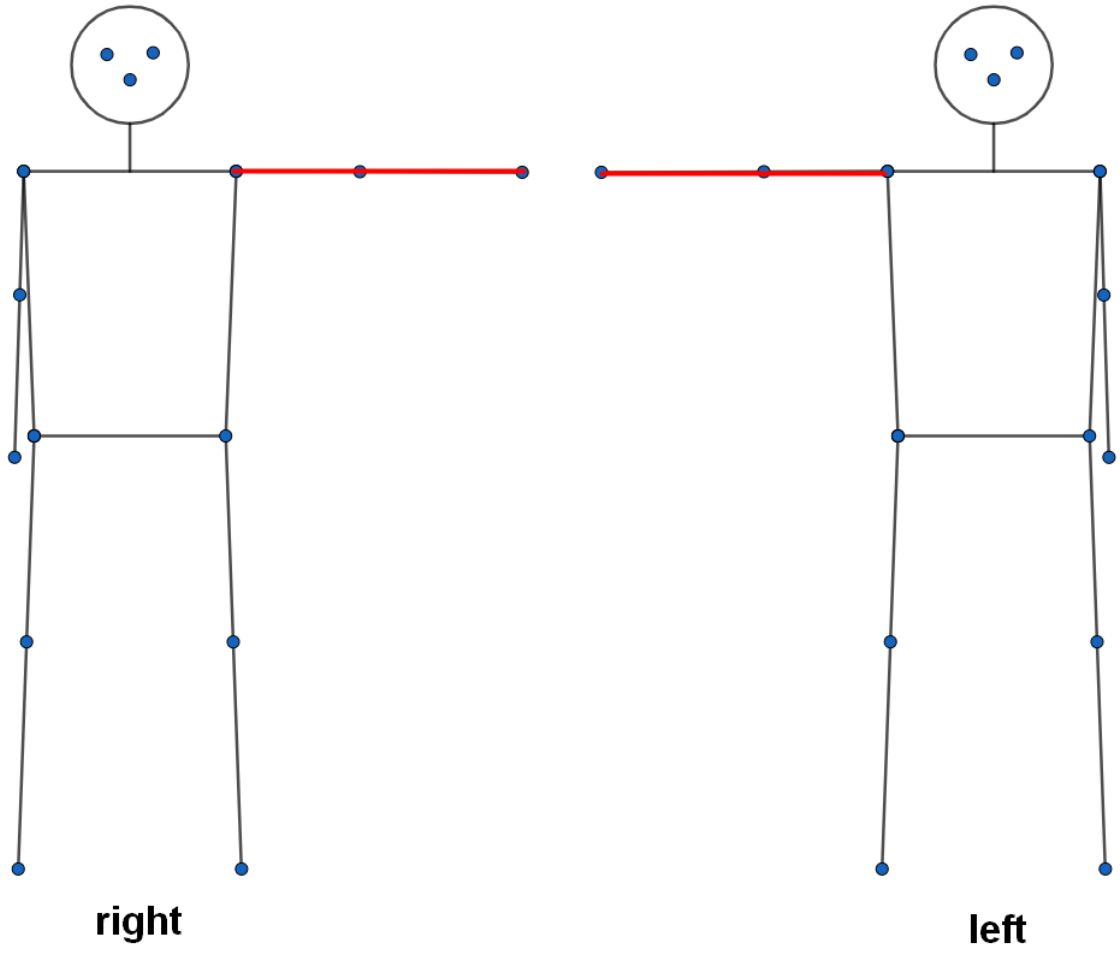


Control pans to the left. Since the robot is opposed to the human body, the human body's right hand shoulder, elbow and wrist joint point position information is used to distinguish the instruction for the robot to translate to the left. When the right hand is raised to a horizontal state, the robot translates to the left. check_line is used to determine whether the coordinates of the three points are on the same straight line. check_distance is used to determine whether the three points are too close. The limit parameter represents the error limit value. The limit value can be modified according to the actual effect.

```
def control_left(shoulder, elbow, wrist):
    global g_control_count, g_left_count
    result = 0
```

```
    state = check_line(shoulder[1], elbow[1], wrist[1], limit=50)
    if state == 1:
        state = check_distance(shoulder[0], elbow[0], wrist[0], limit=50)
    if state == 1:
        g_left_count = g_left_count + 1
        if g_left_count > 3:
            g_bot.left(g_step)
            g_control_count = 10
            result = 1
            clear_action_count()
    return result
```

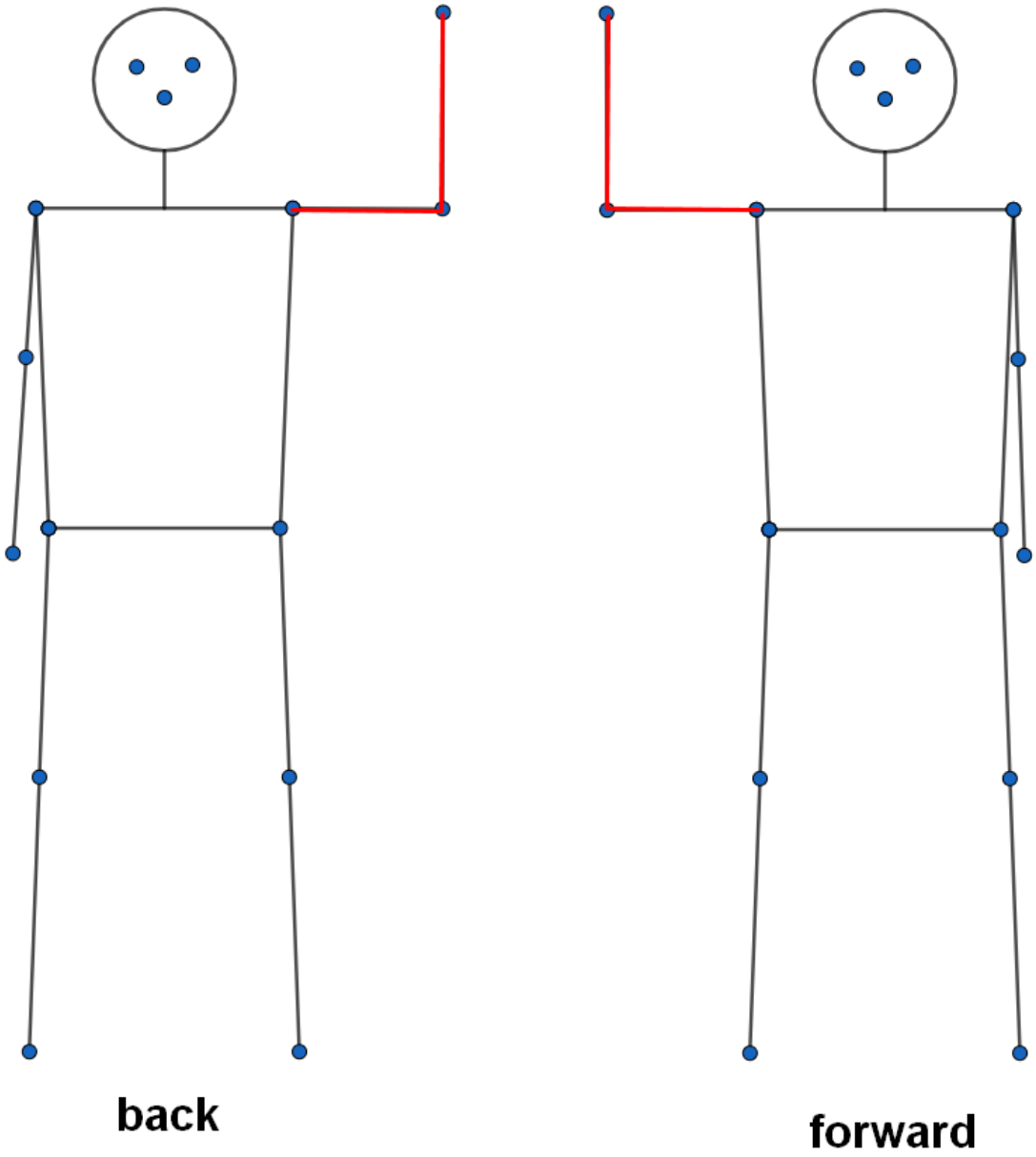
The judgment idea of controlling the right translation function is the same as that of controlling the left translation function. You can control the left translation of the robot by raising your left hand to a horizontal state.



Control forward movement. When the right hand is raised, the robot moves forward. offset_se represents the deviation relationship between the shoulder and the elbow. It is necessary to ensure that the shoulder and elbow are on the same horizontal plane. offset_ew represents the deviation relationship between the elbow and the wrist. It is necessary to ensure that the elbow and wrist are on the same vertical plane. The judgment range of the deviation value can be modified according to the actual effect to optimize the action recognition accuracy.

```
def control_forward(shoulder, elbow, wrist):
    global g_control_count, g_forward_count
    result = 0
    offset_se_x = shoulder[0] - elbow[0]
    offset_se_y = shoulder[1] - elbow[1]
    offset_ew_x = elbow[0] - wrist[0]
    offset_ew_y = elbow[1] - wrist[1]
    if (abs(offset_se_x) > 30 and abs(offset_se_y) < 50 and
        abs(offset_ew_x) < 50 and abs(offset_ew_y) > 30):
        g_forward_count = g_forward_count + 1
        if g_forward_count > 3:
            result = 1
            g_bot.forward(g_step)
            g_control_count = 10
            clear_action_count()
    return result
```

Control the backward movement by raising your left hand.



7.3、Program execution and operation

Note that the following commands need to be run in the system desktop terminal. You can connect the display or use VNC to remotely log in to the desktop to operate.

Open the terminal on the system desktop and run the following command

```
python3 ~/muto/Samples/Deep_Learning/posenet_movement.py
```

At this time, the program starts running, and a camera screen pops up on the desktop. The human body enters the camera screen. When the right hand is raised, the robot is controlled to move forward. When the left hand is raised, the robot is controlled to move backward. When the left hand is stretched out to the left, forming a triangle with the body. , control the robot to rotate right.

If you need to exit the program, press Ctrl+C in the terminal interface to exit the program.