Gesture Control Robotic Arm Action Group

Orin board users can directly open the terminal and input the tutorial commands to run directly. Jetson-Nano board users need to enter the docker container first, then input the tutorial commands in the docker to start the program.

1. Introduction

The gesture control robotic arm action group function is based on gesture recognition, adding the capability of specific gestures to control the robotic arm. When gestures 1-5 appear in the camera image, the robotic arm will execute corresponding actions.

Recognizable gestures include: [One, Two, Three, Four, Five], a total of 5 categories.

2. Launch

- Open the desktop terminal and enter the following command to start the program

```
ros2 run dofbot_pro_mediapipe 11_GestureAction
```

Press the q key in the image or press Ctrl+c in the terminal to exit the program.

3. Source Code

Code path:

```
# Jetson-Nano users need to enter the docker container to view
~/dofbot_pro_ws/src/dofbot_pro_mediapipe/dofbot_pro_mediapipe/11_GestureAction.p
y
```

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
import cv2 as cv
import time
import threading
from dofbot_utils.fps import FPS
from dofbot_utils.robot_controller import Robot_Controller
from dofbot_utils.GestureRecognition import handDetector
from Arm_Lib import Arm_Device

class GestureActionNode(Node):
    def __init__(self):
        super().__init__('gesture_action_node')
        self.hand_detector = handDetector(detectorCon=0.75)
        self.pTime = 0

        # Define the state of grabbing blocks
        self.one_grabbed = 0
        self.two_grabbed = 0
        self.three_grabbed = 0
        self.four_grabbed = 0

        self.block_num = 0

        # Define the number of gesture recognitions
        self.Count_One = 0
        self.Count_Two = 0
        self.Count_Three = 0
        self.Count_Four = 0
        self.Count_Five = 0

        self.arm = Arm_Device()
        self.move_state = False
        self.fps = FPS()
        self.robot = Robot_Controller()
        self.grap_joint = self.robot.get_gripper_value(1)
        self._joint_5 = self.robot.joint5
        self.arm.Arm_serial_servo_write6_array(self.robot.P_LOOK_AT, 1000)

        # OpenCV Video Capture
        self.capture = cv.VideoCapture(0, cv.CAP_V4L2)
        self.capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
        self.capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
        self.get_logger().info(f"Capture get FPS:
{self.capture.get(cv.CAP_PROP_FPS)}")

        timer_period = 0.1  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        ret, frame = self.capture.read()
        if not ret:
            self.get_logger().error("Failed to capture image")
            return

        frame = self.process(frame)

        cv.imshow('frame', frame)
        if cv.waitKey(1) & 0xFF == ord('q'):
            self.get_logger().info("Exiting...")
            cv.destroyAllWindows()
            rclpy.shutdown()

    def process(self, frame):
        frame, lmList = self.hand_detector.findHands(frame, draw=False)
        if len(lmList) != 0:
            gesture = self.hand_detector.get_gesture()
            # print("gesture = {}".format(gesture))

            if gesture == 'One':
                cv.putText(frame, gesture, (250, 30), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 1)
                self.Count_One = self.Count_One + 1
                self.reset_counts('One')
                if self.Count_One >= 10 and self.move_state == False:
                    self.move_state = True
                    self.Count_One = 0
                    self.get_logger().info(f"start arm_ctrl_threading =
{gesture}")
                    task = threading.Thread(target=self.arm_ctrl_threading,
name="arm_ctrl_threading", args=(gesture, ))
                    task.setDaemon(True)
                    task.start()

            elif gesture == 'Two':
                cv.putText(frame, gesture, (250, 30), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 1)
                self.Count_Two = self.Count_Two + 1
                self.reset_counts('Two')
                if self.Count_Two >= 10 and self.move_state == False:
                    self.move_state = True
                    self.Count_Two = 0
                    self.get_logger().info(f"start arm_ctrl_threading =
{gesture}")
                    task = threading.Thread(target=self.arm_ctrl_threading,
name="arm_ctrl_threading", args=(gesture, ))
                    task.setDaemon(True)
                    task.start()

            elif gesture == 'Three':
                cv.putText(frame, gesture, (250, 30), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 1)
                self.Count_Three = self.Count_Three + 1
                self.reset_counts('Three')
                if self.Count_Three >= 10 and self.move_state == False:
                    self.move_state = True
                    self.Count_Three = 0
                    self.get_logger().info(f"start arm_ctrl_threading =
{gesture}")
                    task = threading.Thread(target=self.arm_ctrl_threading,
name="arm_ctrl_threading", args=(gesture, ))
                    task.setDaemon(True)
                    task.start()

            elif gesture == 'Four':
                cv.putText(frame, gesture, (250, 30), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 1)
                self.Count_Four = self.Count_Four + 1
                self.reset_counts('Four')
                if self.Count_Four >= 10 and self.move_state == False:
                    self.move_state = True
                    self.Count_Four = 0
                    self.get_logger().info(f"start arm_ctrl_threading =
{gesture}")
                    task = threading.Thread(target=self.arm_ctrl_threading,
name="arm_ctrl_threading", args=(gesture, ))
                    task.setDaemon(True)
                    task.start()

            elif gesture == 'Five':
                cv.putText(frame, gesture, (250, 30), cv.FONT_HERSHEY_SIMPLEX,
0.9, (0, 255, 0), 1)
                self.Count_Five = self.Count_Five + 1
                self.reset_counts('Five')
                if self.Count_Five >= 10 and self.move_state == False:
                    self.move_state = True
                    self.Count_Five = 0
                    self.get_logger().info(f"start arm_ctrl_threading =
{gesture}")
                    task = threading.Thread(target=self.arm_ctrl_threading,
name="arm_ctrl_threading", args=(gesture, ))
                    task.setDaemon(True)
                    task.start()

        self.fps.update_fps()
        self.fps.show_fps(frame)
        return frame

    def reset_counts(self, gesture):
        if gesture != 'One':
            self.Count_One = 0
        if gesture != 'Two':
            self.Count_Two = 0
        if gesture != 'Three':
            self.Count_Three = 0
        if gesture != 'Four':
            self.Count_Four = 0
        if gesture != 'Five':
            self.Count_Five = 0

    def arm_ctrl_threading(self, gesture):
        self.get_logger().info(f"arm_ctrl_threading gesture = {gesture}")
        if gesture == 'One':
            self.arm.Arm_serial_servo_write6_array(self.robot.P_ACTION_1, 1000)
            time.sleep(1.5)
            self.arm.Arm_serial_servo_write6_array(self.robot.P_LOOK_AT, 1000)
            time.sleep(1)
        elif gesture == 'Two':
            self.arm.Arm_serial_servo_write6_array(self.robot.P_ACTION_2, 1000)
            time.sleep(1.5)
            for i in range(5):
                self.arm.Arm_serial_servo_write(6, 180, 100)
                time.sleep(0.15)
                self.arm.Arm_serial_servo_write(6, 30, 100)
                time.sleep(0.15)
            self.arm.Arm_serial_servo_write6_array(self.robot.P_LOOK_AT, 1000)
            time.sleep(1)
        elif gesture == 'Three':
            for i in range(3):
                self.arm.Arm_serial_servo_write6_array(self.robot.P_ACTION_3,
1200)
                time.sleep(1.2)
                self.arm.Arm_serial_servo_write6_array(self.robot.P_LOOK_AT,
1000)
                time.sleep(1)
        elif gesture == 'Four':
            self.arm.Arm_serial_servo_write6_array(self.robot.P_ACTION_4, 1500)
            time.sleep(1.4)
            for i in range(3):
                self.arm.Arm_serial_servo_write(4, -15, 300)
                time.sleep(0.4)
```

```python
                self.arm.Arm_serial_servo_write(4, 20, 300)
                time.sleep(0.4)
            self.arm.Arm_serial_servo_write6_array(self.robot.P_LOOK_AT, 1000)
            time.sleep(1)
        elif gesture == 'Five':
            for i in range(5):
                self.arm.Arm_serial_servo_write(5, 60, 300)
                time.sleep(0.4)
                self.arm.Arm_serial_servo_write(5, 120, 300)
                time.sleep(0.4)
            self.arm.Arm_serial_servo_write(5, 90, 300)
            time.sleep(0.4)
            self.arm.Arm_serial_servo_write6_array(self.robot.P_LOOK_AT, 1000)
            time.sleep(1)
        self.move_state = False


def main(args=None):
    rclpy.init(args=args)
    node = GestureActionNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
    cv.destroyAllWindows()


if __name__ == '__main__':
    main()
```