Arm Posture Control Robotic Arm

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 arm posture control robotic arm function is based on posture detection, adding the capability of specific postures to control the robotic arm.

Recognizable postures include: [Triangle, Akimbo, Both Hands Up, Left Hand Up, Right Hand Up], 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 12_PoseArm
```

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/12_PoseArm.py
```

```
#!/usr/bin/env python3
# encoding: utf-8
import os
import threading
import cv2 as cv
import numpy as np
import mediapipe as mp
from dofbot_utils.robot_controller import Robot_Controller
from dofbot_utils.fps import FPS
from time import sleep, time

import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

class PoseCtrlArmNode(Node):

    def __init__(self):
        super().__init__('pose_ctrl_arm_node')
```

```
        self.robot = Robot_Controller()
        self.start_action = False
        self.reset_pose()
        self.initHolistic()

        self.bridge = CvBridge()
        self.publisher_ = self.create_publisher(Image, 'processed_image', 10)

        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"Camera FPS:
{self.capture.get(cv.CAP_PROP_FPS)}")

        self.timer = self.create_timer(0.1, self.timer_callback)

    def initHolistic(self, staticMode=False, landmarks=True, detectionCon=0.5,
trackingCon=0.5):
        self.mpHolistic = mp.solutions.holistic
        self.mpFaceMesh = mp.solutions.face_mesh
        self.mpHands = mp.solutions.hands
        self.mpPose = mp.solutions.pose
        self.mpDraw = mp.solutions.drawing_utils
        self.mpholistic = self.mpHolistic.Holistic(
            static_image_mode=staticMode,
            smooth_landmarks=landmarks,
            min_detection_confidence=detectionCon,
            min_tracking_confidence=trackingCon)
        self.lmDrawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 0,
255), thickness=-1, circle_radius=3)
        self.drawSpec = mp.solutions.drawing_utils.DrawingSpec(color=(0, 255,
0), thickness=2, circle_radius=2)

    def findHolistic(self, frame, draw=True):
        poseptArray = []
        lhandptArray = []
        rhandptArray = []
        h, w, c = frame.shape
        img_RGB = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        self.results = self.mpholistic.process(img_RGB)
        if self.results.pose_landmarks:
            if draw: self.mpDraw.draw_landmarks(frame,
self.results.pose_landmarks, self.mpPose.POSE_CONNECTIONS, self.lmDrawSpec,
self.drawSpec)
            for id, lm in enumerate(self.results.pose_landmarks.landmark):
                poseptArray.append([id, lm.x * w, lm.y * h, lm.z])
        if self.results.left_hand_landmarks:
            if draw: self.mpDraw.draw_landmarks(frame,
self.results.left_hand_landmarks, self.mpHands.HAND_CONNECTIONS,
self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.left_hand_landmarks.landmark):
                lhandptArray.append([id, lm.x * w, lm.y * h, lm.z])
        if self.results.right_hand_landmarks:
            if draw: self.mpDraw.draw_landmarks(frame,
self.results.right_hand_landmarks, self.mpHands.HAND_CONNECTIONS,
self.lmDrawSpec, self.drawSpec)
            for id, lm in enumerate(self.results.right_hand_landmarks.landmark):
```

```
                rhandptArray.append([id, lm.x * w, lm.y * h, lm.z])
        return frame, poseptArray, lhandptArray, rhandptArray

    def process(self, frame):
        frame = cv.flip(frame, 1)
        frame, pointArray, lhandptArray, rhandptArray = self.findHolistic(frame)
        if self.start_action == False:
            self.start_action = True
            threading.Thread(target=self.arm_ctrl_threading, args=(pointArray,
lhandptArray, rhandptArray)).start()
        return frame

    def get_angle(self, v1, v2):
        angle = np.dot(v1, v2) / (np.sqrt(np.sum(v1 * v1)) * np.sqrt(np.sum(v2 *
v2)))
        angle = np.arccos(angle) / 3.14 * 180
        cross = v2[0] * v1[1] - v2[1] * v1[0]
        if cross < 0:
            angle = - angle
        return angle

    def get_pos(self, keypoints):
        str_pose = ""
        # Calculate the angle between left arm and horizontal direction
        keypoints = np.array(keypoints)
        v1 = keypoints[12] - keypoints[11]
        v2 = keypoints[13] - keypoints[11]
        angle_left_arm = self.get_angle(v1, v2)
        # Calculate the angle between right arm and horizontal direction
        v1 = keypoints[11] - keypoints[12]
        v2 = keypoints[14] - keypoints[12]
        angle_right_arm = self.get_angle(v1, v2)
        # Calculate the angle of left elbow
        v1 = keypoints[11] - keypoints[13]
        v2 = keypoints[15] - keypoints[13]
        angle_left_elow = self.get_angle(v1, v2)
        # Calculate the angle of right elbow
        v1 = keypoints[12] - keypoints[14]
        v2 = keypoints[16] - keypoints[14]
        angle_right_elow = self.get_angle(v1, v2)

        if 90 < angle_left_arm < 120 and -120 < angle_right_arm < -90:
            str_pose = ""
        elif 90 < angle_left_arm < 120 and 90 < angle_right_arm < 120:
            # Left hand down, right hand up
            str_pose = "RIGHT_UP"
        elif -120 < angle_left_arm < -90 and -120 < angle_right_arm < -90:
            # Right hand down, left hand up
            str_pose = "LEFT_UP"
        elif -120 < angle_left_arm < -90 and 90 < angle_right_arm < 120:
            # Both hands up
            str_pose = "ALL_HANDS_UP"
        elif 130 < angle_left_arm < 150 and -150 < angle_right_arm < -130 and 90
< angle_left_elow < 120 and -120 < angle_right_elow < 90:
            # Hands on waist
            str_pose = "AKIMBO"
        elif -150 < angle_left_arm < -120 and 120 < angle_right_arm < 150 and
-85 < angle_left_elow < -55 and 55 < angle_right_elow < 85:
```

```
            # Both hands form triangle
            str_pose = "TRIANGLE"
        print("str_pose = ", str_pose)
        print("angle_left_arm = ", angle_left_arm, "\tangle_right_arm = ",
angle_right_arm)
        print("angle_left_elow = ", angle_left_elow, "\tangle_right_elow = ",
angle_right_elow)
        return str_pose

    def arm_ctrl_threading(self, pointArray, lhandptArray, rhandptArray):
        keypoints = ['' for i in range(33)]
        if len(pointArray) != 0:
            for i in range(len(pointArray)):
                keypoints[i] = (pointArray[i][1], pointArray[i][2])
            str_pose = self.get_pos(keypoints)
            if str_pose:
                print("str_pose = ", str_pose)
            if str_pose == "RIGHT_UP":
                self.RIGHT_UP()
            elif str_pose == "LEFT_UP":
                self.LEFT_UP()
            elif str_pose == "ALL_HANDS_UP":
                self.ALL_HANDS_UP()
            elif str_pose == "TRIANGLE":
                self.TRIANGLE()
            elif str_pose == "AKIMBO":
                self.AKIMBO()
        self.start_action = False

    def reset_pose(self):
        self.robot.arm_move_6(self.robot.P_POSE_INIT, 1000)
        sleep(1.5)

    def RIGHT_UP(self):
        self.robot.arm_move_6(self.robot.P_RIGHT_UP, 1000)
        sleep(3)
        self.reset_pose()

    def LEFT_UP(self):
        self.robot.arm_move_6(self.robot.P_LEFT_UP, 1000)
        sleep(3)
        self.reset_pose()

    def ALL_HANDS_UP(self):
        self.robot.arm_move_6(self.robot.P_HANDS_UP, 1000)
        sleep(3)
        self.reset_pose()

    def TRIANGLE(self):
        self.robot.arm_move_6([90, 131, 52, 0, 90, 180], 1500)
        sleep(1.5)
        self.robot.arm_move_6([45, 180, 0, 0, 90, 180], 1500)
        sleep(2)
        self.robot.arm_move_6([135, 180, 0, 0, 90, 180], 1500)
        sleep(2)
        self.robot.arm_move_6([90, 131, 52, 0, 90, 180], 1500)
        sleep(2)
        self.reset_pose()
```

```
    def AKIMBO(self):
        for i in range(3):
            self.robot.arm_move_6(self.robot.P_ACTION_3, 1200)
            sleep(1.2)
            self.robot.arm_move_6(self.robot.P_LOOK_AT, 1000)
            sleep(1)
        self.reset_pose()

    def timer_callback(self):
        ret, frame = self.capture.read()
        if ret:
            frame = self.process(frame)
            processed_image_msg = self.bridge.cv2_to_imgmsg(frame, "bgr8")
            self.publisher_.publish(processed_image_msg)
            cv.imshow('frame', frame)
            if cv.waitKey(1) & 0xFF == ord('q'):
                self.capture.release()
                cv.destroyAllWindows()
                rclpy.shutdown()


def main(args=None):
    rclpy.init(args=args)
    pose_ctrl_arm_node = PoseCtrlArmNode()
    rclpy.spin(pose_ctrl_arm_node)
    pose_ctrl_arm_node.destroy_node()
    rclpy.shutdown()

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