Arm gesture control robot

1. Introduction

The arm gesture control robot function is based on gesture detection, adding the function of specific gesture control robot.

The recognizable gestures are: [triangle, akimbo, raised hands, raised left hand, raised right hand], a total of 5 categories.

2. Start

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

```
rosrun dofbot_mediapipe 12_PoseArm.py
```

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

3. Source code

Code path:

```
~/dofbot_ws/src/dofbot_mediapipe/scripts/12_PoseArm.py
```

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


class PoseCtrlArm:

    def __init__(self):

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

    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 = ""
        # 计算左臂与水平方向的夹角
        # Calculate the angle between the left arm and the horizontal
        keypoints = np.array(keypoints)
        v1 = keypoints[12] - keypoints[11]
        v2 = keypoints[13] - keypoints[11]
        angle_left_arm = self.get_angle(v1, v2)
        # 计算右臂与水平方向的夹角
        # Calculate the angle between the right arm and the horizontal direction
        v1 = keypoints[11] - keypoints[12]
        v2 = keypoints[14] - keypoints[12]
        angle_right_arm = self.get_angle(v1, v2)
        # 计算左肘的夹角
        # Calculate the angle of the left elbow
        v1 = keypoints[11] - keypoints[13]
        v2 = keypoints[15] - keypoints[13]
        angle_left_elow = self.get_angle(v1, v2)
        # 计算右肘的夹角
        # Calculate the angle of the 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:
            # 左手放下，举起右手
            # Put your left hand down and raise your right hand
            str_pose = "RIGHT_UP"
        elif -120<angle_left_arm<-90 and -120<angle_right_arm<-90:
            # 右手放下，举起左手
            # Put your right hand down and raise your left hand
            str_pose = "LEFT_UP"
        elif -120<angle_left_arm<-90 and 90<angle_right_arm<120:
            # 手上向上 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:
            # 双手叉腰 Hands on hips
            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:
            # 双手合成三角形 Make a triangle with both hands
            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()

if __name__ == '__main__':
    pose_ctrl_arm = PoseCtrlArm()
    capture = cv.VideoCapture(0)
    # capture.set(6, cv.VideoWriter.fourcc('M', 'J', 'P', 'G'))
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    print("capture get FPS : ", capture.get(cv.CAP_PROP_FPS))
    fps = FPS()
    while capture.isOpened():
        ret, frame = capture.read()
        fps.update_fps()
        frame = pose_ctrl_arm.process(frame)
        if cv.waitKey(1) & 0xFF == ord('q'): break
        fps.show_fps(frame)
        cv.imshow('frame', frame)
    capture.release()
    cv.destroyAllWindows()
```