Miroir Pepper : P24AB04

Note d'application

Intégration de Mediapipe dans l'environnement ROS Melodic

Développeuse : Fatima Elkhadiri

Tuteur Académique et client : M. Sébastien Lengagne

Année-académique : 2024 - 2025

Table des matières

I. Introduction

Dans le cadre de ce projet, nous explorons l'intégration de MediaPipe au sein de ROS Melodic afin d’analyser et de traiter des mouvements humains pour leur reproduction sur les robots Pepper et Braccio. Cependant, cette implémentation présente plusieurs défis techniques, notamment l’incompatibilité de Python 3 avec ROS Melodic, les conflits de dépendances avec OpenCV, ainsi que la gestion des environnements virtuels pour assurer une exécution stable du programme. Pour pallier ces problèmes, nous avons opté pour l'utilisation d'Anaconda, qui permet de créer un environnement Python 3 isolé, évitant ainsi les conflits avec les dépendances de ROS. Ce rapport détaille la solution adoptée et le test réalisé pour garantir une interaction fluide entre MediaPipe et ROS.

II. Intégration de Mediapipe

A. Téléchargement et exécustion du script d'installation d'**Anaconda**

- Ouvrez un terminal.
- Téléchargez le script depuis le site officiel d'Anaconda :

```
wget https://repo.anaconda.com/archive/Anaconda3-2023.03-1-Linux-x86_64.sh
```

- Lancez le script téléchargé :

```
bash Anaconda3-2023.07-1-Linux-x86_64.sh
```

Suivez les instructions affichées :

- Appuyez sur **Entrée** pour continuer.
- Lisez et acceptez le contrat de licence en tapant **yes**.
- Choisissez le répertoire d'installation (par défaut `~/anaconda3`).

Ajouter Anaconda à votre PATH

Lors de l'installation, on vous demandera si vous souhaitez ajouter Anaconda à votre `PATH`. Répondez **yes** pour qu'il soit automatiquement ajouté.

- Si vous avez oublié, vous pouvez ajouter manuellement la ligne suivante à votre fichier `~/.bashrc` :

```
export PATH="$HOME/anaconda3/bin:$PATH"
source ~/.bashrc
```

Note : il faut veiller à ce que l’espace du disque soit suffisant pour l’installation d’Anaconda

B. Création d'environnement Anaconda

```
conda create -n ros_mediapipe python=3.11 -y

conda activate ros_mediapipe

pip install mediapipe

pip install opencv-python
```

- Ajoutez le chemin de votre environnement Conda Python 3 dans le `PYTHONPATH` pour que ROS utilise cet environnement :

```
export
PYTHONPATH=/chemin/vers/conda/envs/ros_mediapipe/lib/python3.11/site-
packages:$PYTHONPATH
```

- Puis :

```
source ~/.bashrc
```

C. Création du workspace

- Créez un dossier pour ton espace de travail :

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws
catkin_make
```

- Ajoutez l'espace de travail au `bashrc` :

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

1. Exemple complet de `CMakeLists.txt`

Exemple de fichier `CMakeLists.txt` pour un package ROS :

```
cmake_minimum_required(VERSION 3.0.2)
project(mediapipe_ros)

## Trouver les dépendances nécessaires
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
)

## Inclure les répertoires d'en-tête
catkin_package()

## Installer les scripts Python (etape D - 4)
catkin_install_python(PROGRAMS
    scripts/mediapipe_node.py
    DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

## Ajouter des dépendances si nécessaires
include_directories(
  ${catkin_INCLUDE_DIRS}
)
```

D. Création du nœud ROS Python utilisant MediaPipe

Créez un package ROS et un nœud Python qui utilise MediaPipe. Assurez-vous de bien activer l'environnement Conda avant d'exécuter votre nœud. (`conda activate ros_mediapipe`)

1. **Créez le package ROS** :

   ```
   cd ~/catkin_ws/src
   catkin_create_pkg mediapipe_ros rospy std_msgs sensor_msgs cv_bridge
   ```

2. **Ajoutez votre script Python dans le package** : Placez votre programme Python dans le dossier `scripts` du package, par exemple :

   mkdir -p ~/catkin_ws/src/mediapipe_ros/scripts

   touch ~/catkin_ws/src/mediapipe_ros/scripts/mediapipe_node.py

3. **Rendez le script exécutable** :

   ```
   chmod +x ~/catkin_ws/src/mediapipe_ros/scripts/mediapipe_node.py
   ```

4. **Modifiez `CMakeLists.txt` pour inclure les scripts Python** : Dans `~/catkin_ws/src/mediapipe_ros/CMakeLists.txt`, ajoutez :

   ```
   catkin_install_python(PROGRAMS
       scripts/mediapipe_node.py
       DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
   )
   ```

E. Le nœud ROS mediapipe_node.py

Le script à écrire dans ce nœud est **dans l'annexe 1**

F. Compiler et Lancer le Noeud

- Reviens à la racine de ton workspace :

```
cd ~/catkin_ws
catkin_make
source devel/setup.bash
```

- Vérifie si ROS tourne, exécutez dans un nouveau terminal :

```
Roscore
```

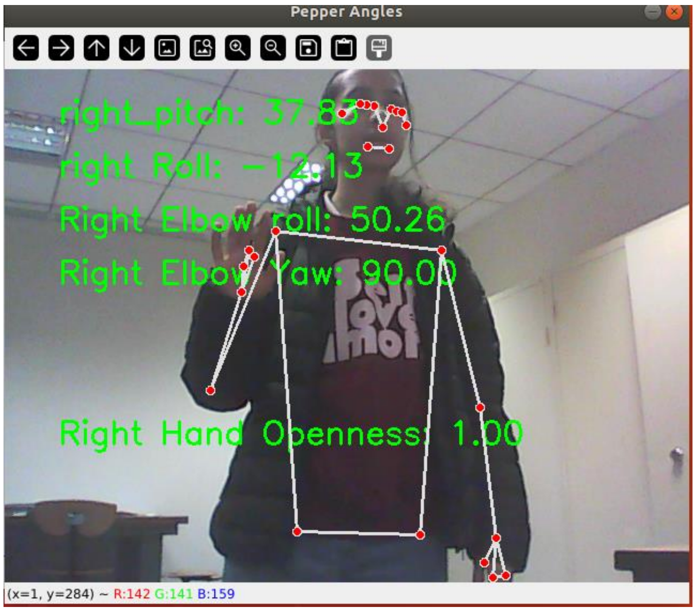
- Puis :

```
rosrun mediapipe_ros mediapipe_node.py
```

- Vérifie si la caméra est accessible :

```
ls /dev/video*
```

III. Résultat :



```
right_pitch: 37.83
right Roll: -12.13
Right Elbow roll: 50.26
Right Elbow Yaw: 90.00
Right Hand Openness: 1.00
```

Cette image montre la sortie du code présentant les angles du bras droit, avec une squelettisation des articulations et l'affichage en temps réel des angles et de l'ouverture de la main.

IV. Conclusion

L'intégration de MediaPipe avec ROS Melodic a nécessité l'utilisation d'Anaconda pour gérer les incompatibilités et stabiliser l'exécution. Les ajustements réalisés ont permis une interaction fluide, assurant une reconnaissance précise des mouvements et ouvrant la voie à de futures optimisations.

V. Annexes

A. Annexe 1 : mediapipe_node.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import rospy
from std_msgs.msg import Float64MultiArray
import cv2
import mediapipe as mp
import numpy as np
import math

rospy.init_node('pepper_angles_publisher', anonymous=True)
pub = rospy.Publisher('/robot/ikine_solution', Float64MultiArray, queue_size=10)



mp_pose = mp.solutions.pose
pose = mp_pose.Pose()

mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=2,
min_detection_confidence=0.5)

cap = cv2.VideoCapture(0, cv2.CAP_GSTREAMER)

mp_drawing = mp.solutions.drawing_utils

def init_kalman_filter():
    kalman = cv2.KalmanFilter(2, 1)
    kalman.measurementMatrix = np.array([[1, 0]], np.float32)
    kalman.transitionMatrix = np.array([[1, 1], [0, 1]], np.float32)
    kalman.processNoiseCov = np.array([[1, 0], [0, 1]], np.float32) * 1e-5
    kalman.measurementNoiseCov = np.array([[1]], np.float32) * 1e-2
    kalman.errorCovPost = np.array([[1, 0], [0, 1]], np.float32)
    kalman.statePost = np.array([[0], [0]], np.float32)
    return kalman
'''
def is_hand_open(landmarks):
    # ????????
    thumb_tip = landmarks[mp_hands.HandLandmark.THUMB_TIP]
    index_tip = landmarks[mp_hands.HandLandmark.INDEX_FINGER_TIP]

    # ?????????????
    distance = math.sqrt((thumb_tip.x - index_tip.x) ** 2 + (thumb_tip.y - index_tip.y) ** 2)

    # ????????,????0.0?1.0??
    max_distance = 0.1  # ????????,??????????
    openness = min(distance / max_distance, 1.0)  # ?????1.0

    return abs(openness - 0.3)
'''
def is_hand_open(landmarks):
    thumb_tip = landmarks[mp_hands.HandLandmark.THUMB_TIP].x,
landmarks[mp_hands.HandLandmark.THUMB_TIP].y
    index_tip = landmarks[mp_hands.HandLandmark.INDEX_FINGER_TIP].x,
landmarks[mp_hands.HandLandmark.INDEX_FINGER_TIP].y



    distance = math.sqrt((thumb_tip[0] - index_tip[0]) ** 2 + (thumb_tip[1] - index_tip[1]) ** 2)



    if distance > 0.04:
        return 10
    else:
        return 73

def atan2_deg(y, x):
    return math.degrees(math.atan2(y, x))

def calculate_angle_roll_left(v1, v2):

    dot_product = v1[0] * v2[0] + v1[1] * v2[1]

    magnitude_v1 = math.sqrt(v1[0] ** 2 + v1[1] ** 2)
    magnitude_v2 = math.sqrt(v2[0] ** 2 + v2[1] ** 2)

    if magnitude_v1 == 0 or magnitude_v2 == 0:
        return 0.0

    angle = math.acos(dot_product / (magnitude_v1 * magnitude_v2)) * (180.0 / math.pi)

    #if v1[1] < 0:

        #angle = 180

    return angle

def calculate_angle_roll_right(v1, v2):

    dot_product = v1[0] * v2[0] + v1[1] * v2[1]

    magnitude_v1 = math.sqrt(v1[0] ** 2 + v1[1] ** 2 )
    magnitude_v2 = math.sqrt(v2[0] ** 2 + v2[1] ** 2)

    if magnitude_v1 == 0 or magnitude_v2 == 0:
        return 0.0
    angle = math.acos(dot_product / (magnitude_v1 * magnitude_v2)) * (180.0 / math.pi)

    #if v1[1] < 0:

        #angle = 180

    return -angle

def calculate_angle_pitch_left(v1, v2):

    magnitude_v1 = math.sqrt(v1[0] ** 2 + v1[1] ** 2 + v1[2] ** 2)
    magnitude_v2 = math.sqrt(v2[0] ** 2 + v2[1] ** 2 + v2[2] ** 2)

    if magnitude_v1 == 0 or magnitude_v2 == 0:
        return 0.0

    dot_product = v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2]

    cross_product = np.cross(v1, v2)

    angle_radians = math.atan2(np.linalg.norm(cross_product), dot_product)

    pitch = np.degrees(angle_radians) - 15

    if v1[1] < 0:

        pitch = -pitch - 40

    return pitch

def calculate_angle_pitch_right(v1, v2):
    magnitude_v1 = math.sqrt(v1[0] ** 2 + v1[1] ** 2 + v1[2] ** 2)
    magnitude_v2 = math.sqrt(v2[0] ** 2 + v2[1] ** 2 + v2[2] ** 2)

    if magnitude_v1 == 0 or magnitude_v2 == 0:
        return 0.0

    dot_product = v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2]

    cross_product = np.cross(v1, v2)

    angle_radians = math.atan2(np.linalg.norm(cross_product), dot_product)

    pitch = np.degrees(angle_radians)

    if v1[1] < 0:

        pitch = -pitch - 40

    return pitch

def calculate_elbow_angle(shoulder, elbow, wrist):
    # ????
    vec_se = shoulder - elbow  # ?????????
    vec_ew = wrist - elbow     # ?????????

    # ?????
    vec_se_normalized = vec_se / np.linalg.norm(vec_se)
    vec_ew_normalized = vec_ew / np.linalg.norm(vec_ew)

    # ???????????
    angle = np.arccos(np.dot(vec_se_normalized, vec_ew_normalized))
    angle_degrees = np.degrees(angle)  # ???????

    # ????
    elbow_angle = 180 - angle_degrees

    # ????????????
    wrist_elbow_distance = np.linalg.norm(vec_ew)
    shoulder_elbow_distance = np.linalg.norm(vec_se)

    # ????????????????????????,??????????1/10
    if wrist_elbow_distance < (shoulder_elbow_distance / 2):
        elbow_angle = elbow_angle / 10

    return elbow_angle


# Function to adjust elbow angle for display
def adjust_elbow_angle(elbow_angle, side):
    if side == 'left':
        # Left elbow should range from -89.5° to 0°
        if elbow_angle > 90:
            return -89.5
        return -89.5-(-(90 - elbow_angle))  # Convert to the range [-89.5, 0]

    elif side == 'right':
        # Right elbow should range from 0° to 89.5°
        if elbow_angle > 90:
            return 89.5
        return 89.5-(90 - elbow_angle)  # Convert to the range [0, 89.5]

def calculate_elbow_yaw_left(shoulder, elbow, wrist):
    # ????
```

```
v_upper = elbow - shoulder
v_upper_norm = np.linalg.norm(v_upper)
v_upper_normalized = v_upper / v_upper_norm

# ????
v_forearm = wrist - elbow

# ????????
up_vector = np.array([0, 1, 0])

# ??????????????
if np.allclose(np.abs(np.dot(v_upper_normalized, up_vector)), 1.0):
    # ????,???????????
    up_vector = np.array([0, 0, 1])

# ?? Z ?:????????,??????????
z_axis = up_vector - np.dot(up_vector, v_upper_normalized) * v_upper_normalized
z_axis_norm = np.linalg.norm(z_axis)
if z_axis_norm == 0:
    # ????????,????? up_vector
    up_vector = np.array([1, 0, 0])
    z_axis = up_vector - np.dot(up_vector, v_upper_normalized) * v_upper_normalized
    z_axis_norm = np.linalg.norm(z_axis)
z_axis_normalized = z_axis / z_axis_norm

# ?? X ?:???? Z ???,???????
x_axis = np.cross(z_axis_normalized, v_upper_normalized)
x_axis_normalized = x_axis / np.linalg.norm(x_axis)

# ???????????????
v_forearm_proj = v_forearm - np.dot(v_forearm, v_upper_normalized) * v_upper_normalized
v_forearm_proj_norm = np.linalg.norm(v_forearm_proj)
```

```
    if v_forearm_proj_norm == 0:
        # ????????,?????????
        angle_degrees = -90 if np.dot(v_forearm, v_upper_normalized) > 0 else 90
        return angle_degrees

    v_forearm_proj_normalized = v_forearm_proj / v_forearm_proj_norm

    # ????????? X ?? Z ?????
    x_component = np.dot(v_forearm_proj_normalized, x_axis_normalized)
    z_component = np.dot(v_forearm_proj_normalized, z_axis_normalized)

    # ????(??)
    angle_radians = np.arctan2(z_component, x_component)

    # ?????
    angle_degrees = np.degrees(angle_radians)

    # ??????,?????? -90 ?,??? 0 ?,????? 90 ?
    return angle_degrees

def calculate_elbow_yaw_right(shoulder, elbow, wrist):
    # ????
    v_upper = elbow - shoulder
    v_upper_normalized = v_upper / np.linalg.norm(v_upper)

    # ????
    v_forearm = wrist - elbow

    # ?????????????????
    v_forearm_proj = v_forearm - np.dot(v_forearm, v_upper_normalized) * v_upper_normalized
    v_forearm_proj_norm = np.linalg.norm(v_forearm_proj)
```

```
if v_forearm_proj_norm == 0:
    # ????????,?????????
    # ?? yaw ?? 90 ?
    return 90.0

v_forearm_proj_normalized = v_forearm_proj / v_forearm_proj_norm

# ????????
up_vector = np.array([0, 1, 0])

# ???????????????????
up_proj = up_vector - np.dot(up_vector, v_upper_normalized) * v_upper_normalized
up_proj_norm = np.linalg.norm(up_proj)

if up_proj_norm == 0:
    # ????????,???????????
    up_vector = np.array([0, 0, 1])
    up_proj = up_vector - np.dot(up_vector, v_upper_normalized) * v_upper_normalized
    up_proj_norm = np.linalg.norm(up_proj)

up_proj_normalized = up_proj / up_proj_norm

# ??????????????????
dot_product = np.dot(v_forearm_proj_normalized, up_proj_normalized)
angle_radians = np.arccos(np.clip(dot_product, -1.0, 1.0))
angle_degrees = np.degrees(angle_radians)

# ?????? yaw ??
if angle_degrees <= 90:
    yaw_angle_degrees = 90 - angle_degrees
else:
        yaw_angle_degrees = -(angle_degrees - 90)

    return -yaw_angle_degrees

def constrain_angle(angle, min_value, max_value):
    return max(min(angle, max_value), min_value)

def calculate_angle_wrist(vector1, vector2):
    """Calculate the angle between two vectors."""
    unit_vector1 = vector1 / np.linalg.norm(vector1)
    unit_vector2 = vector2 / np.linalg.norm(vector2)
    dot_product = np.dot(unit_vector1, unit_vector2)
    angle = np.arccos(np.clip(dot_product, -1.0, 1.0))  # Clip for numerical stability
    angle = np.degrees(angle)
    return angle

kalman_wrist_roll = init_kalman_filter()

def calculateAngle_right_elbow_roll(elbow, wrist):
    # ??????
    v_forearm = np.array(wrist) - np.array(elbow)

    # Z ??????
    z_axis = np.array([0, 0, -1])



    # ??????
    magnitude_forearm = np.linalg.norm(v_forearm)
    magnitude_z_axis = np.linalg.norm(z_axis)  # ?? 1,?????

    # ????????,?????
    if magnitude_forearm == 0 or magnitude_z_axis == 0:
        return 0.0

    # ????
    dot_product = np.dot(v_forearm, z_axis)

    # ????,?? acos
    angle_radians = math.acos(dot_product / (magnitude_forearm * magnitude_z_axis))

    # ?????
    angle_with_z_axis = np.degrees(angle_radians)

    return angle_with_z_axis

kalman_left_pitch = init_kalman_filter()
kalman_left_roll = init_kalman_filter()
kalman_left_elbow_roll = init_kalman_filter()
kalman_left_elbow_yaw = init_kalman_filter()

kalman_head_yaw = init_kalman_filter()
kalman_head_pitch = init_kalman_filter()

kalman_right_pitch = init_kalman_filter()
kalman_right_roll = init_kalman_filter()
kalman_right_elbow_roll = init_kalman_filter()
kalman_right_elbow_yaw = init_kalman_filter()

while not rospy.is_shutdown():
    ret, frame = cap.read()
    if not ret:
        rospy.logwarn("Failed to capture frame from camera.")
        break
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)


    results = pose.process(rgb_frame)
    results_hands = hands.process(rgb_frame)


    right_pitch = 90
    right_roll = -0.5
    right_elbow_roll = 0.5
    right_elbow_yaw = 90
    left_pitch = 90
    left_roll = 0.5
    left_elbow_roll = -0.5
    left_elbow_yaw = -90
    wrist_roll_angle = 0


    if results_hands.multi_hand_landmarks:



        for hand_landmarks in results_hands.multi_hand_landmarks:
            mp_drawing.draw_landmarks(frame, hand_landmarks,
mp_hands.HAND_CONNECTIONS)


            landmarks_hand = hand_landmarks.landmark


        if results.pose_landmarks:
            landmarks = results.pose_landmarks.landmark
            pinky_base = np.array([landmarks_hand[17].x, landmarks_hand[17].y,
landmarks_hand[17].z])


            # Get the coordinates for left shoulder, elbow, wrist
            left_shoulder = np.array([landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER].x,
                        landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER].y])
            left_elbow = np.array([landmarks[mp_pose.PoseLandmark.LEFT_ELBOW].x,
                 landmarks[mp_pose.PoseLandmark.LEFT_ELBOW].y])
left_wrist = np.array([landmarks[mp_pose.PoseLandmark.LEFT_WRIST].x,
                 landmarks[mp_pose.PoseLandmark.LEFT_WRIST].y])

# Get the coordinates for right shoulder, elbow, wrist
right_shoulder = np.array([landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER].y])
right_elbow = np.array([landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW].y])
right_wrist = np.array([landmarks[mp_pose.PoseLandmark.RIGHT_WRIST].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_WRIST].y])

left_shoulder1 = np.array([landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER].x,
                    landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER].y,
                    landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER].z])
left_elbow1 = np.array([landmarks[mp_pose.PoseLandmark.LEFT_ELBOW].x,
                 landmarks[mp_pose.PoseLandmark.LEFT_ELBOW].y,
                 landmarks[mp_pose.PoseLandmark.LEFT_ELBOW].z])
left_wrist1 = np.array([landmarks[mp_pose.PoseLandmark.LEFT_WRIST].x,
                 landmarks[mp_pose.PoseLandmark.LEFT_WRIST].y,
                 landmarks[mp_pose.PoseLandmark.LEFT_WRIST].z])

right_shoulder1 = np.array([landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER].y,
                    landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER].z])
right_elbow1 = np.array([landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW].x,
                    landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW].y,
                    landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW].z])
right_wrist1 = np.array([
    landmarks_hand[0].x, landmarks_hand[0].y, landmarks_hand[0].z
      ])
    nose = np.array([landmarks[mp_pose.PoseLandmark.NOSE].x,
            landmarks[mp_pose.PoseLandmark.NOSE].y,
            landmarks[mp_pose.PoseLandmark.NOSE].z])

    left_eye = np.array([landmarks[mp_pose.PoseLandmark.LEFT_EYE_OUTER].x,
              landmarks[mp_pose.PoseLandmark.LEFT_EYE_OUTER].y,
              landmarks[mp_pose.PoseLandmark.LEFT_EYE_OUTER].z])

    right_eye = np.array([landmarks[mp_pose.PoseLandmark.RIGHT_EYE_OUTER].x,
              landmarks[mp_pose.PoseLandmark.RIGHT_EYE_OUTER].y,
              landmarks[mp_pose.PoseLandmark.RIGHT_EYE_OUTER].z])

            # Calculate vectors
    vec_ew = right_wrist1 - right_elbow1
    palm_vector = pinky_base - right_wrist1

    eye_center = (left_eye + right_eye) / 2



    head_vector = np.array([eye_center[0] - nose[0], eye_center[1] - nose[1], eye_center[2] -
nose[2]])



    head_yaw = -(atan2_deg(head_vector[1], head_vector[0]) + 90)



    head_pitch = -(atan2_deg(head_vector[2], np.sqrt(head_vector[0]**2 + head_vector[1]**2))
- 50)

    # Define the vertical axis vector starting from the right_wrist1 landmark
    vertical_axis = np.array([0, -1, 0])  # A unit vector pointing downward (negative y-direction)
    # Calculate the wrist roll angle relative to the vertical axis
    wrist_roll_angle = calculate_angle_wrist(vertical_axis, palm_vector)
```

```
        wrist_roll_angle = abs(wrist_roll_angle -180)

        # Calculate left and right elbow bending angles
        left_elbow_angle = calculate_elbow_angle(left_shoulder, left_elbow, left_wrist)
        right_elbow_angle = calculate_elbow_angle(right_shoulder, right_elbow, right_wrist)
        right_elbow_yaw = calculate_elbow_yaw_right(right_shoulder1, right_elbow1,
right_wrist1) * 1.5
        left_elbow_yaw = calculate_elbow_yaw_left(left_shoulder1, left_elbow1, left_wrist1) * 1.3

        left_arm_up = left_elbow - left_shoulder
        right_arm_up = right_elbow - right_shoulder

        left_arm_up1 = left_elbow1 - left_shoulder1
        right_arm_up1 = right_elbow1 - right_shoulder1

        v1 = [0, 0, -1]

        v2 = [-1, 0, 0]
        '''
        y_axis = np.array([0, 1, 0])
        x_axis = np.array([1, 0, 0])



        dot_product_y = np.dot(left_arm_up1, y_axis)
        dot_product_x = np.dot(left_arm_up1, x_axis)
        magnitude_upper_arm = np.linalg.norm(left_arm_up1)



        if magnitude_upper_arm == 0:
            left_pitch = 90
        else:
            angle_with_y = np.degrees(np.arccos(dot_product_y / (magnitude_upper_arm *
np.linalg.norm(y_axis))))

            angle_with_x = np.degrees(np.arccos(dot_product_x / (magnitude_upper_arm *
np.linalg.norm(x_axis))))


        if abs(angle_with_y) < 50:

            left_pitch = 90
        elif abs(angle_with_x) < 50:

            left_pitch = 90
        else:

            left_pitch = calculate_angle_pitch_left(left_arm_up1, v1)
        '''
        #left_pitch = calculate_angle_pitch_left(left_arm_up1, v1)

        left_roll = calculate_angle_roll_left(left_arm_up, v2) - 90

        if abs(left_roll) > 60:

            left_pitch = 0

        else:

            left_pitch = calculate_angle_pitch_left(left_arm_up1, v1)

        left_elbow_roll =adjust_elbow_angle(left_elbow_angle, 'left')
```

```
right_roll = calculate_angle_roll_right(-right_arm_up, v2) + 95

if abs(right_roll) > 60:

    right_pitch = 0

else:

    right_pitch = calculate_angle_pitch_right(right_arm_up1, v1)



if abs(right_roll) > 50 :

    right_elbow_roll =adjust_elbow_angle(right_elbow_angle, 'right')

elif (abs(right_roll) > 2) and (abs(right_roll) < 12) :

    right_elbow_roll =adjust_elbow_angle(right_elbow_angle, 'right') * 1.5



else:

    right_elbow_roll = calculateAngle_right_elbow_roll(right_elbow1, right_wrist1) * 2.7



left_pitch = kalman_left_pitch.correct(np.array([[left_pitch]], np.float32))[0][0]
left_roll = kalman_left_roll.correct(np.array([[left_roll]], np.float32))[0][0]
left_elbow_roll = kalman_left_elbow_roll.correct(np.array([[left_elbow_roll]],
np.float32))[0][0]
left_elbow_yaw = kalman_left_elbow_yaw.correct(np.array([[left_elbow_yaw]],
np.float32))[0][0]

right_pitch = kalman_right_pitch.correct(np.array([[right_pitch]], np.float32))[0][0]
```

```
            right_roll = kalman_right_roll.correct(np.array([[right_roll]], np.float32))[0][0]

            right_elbow_roll = kalman_right_elbow_roll.correct(np.array([[right_elbow_roll]],
np.float32))[0][0]

            right_elbow_yaw = kalman_right_elbow_yaw.correct(np.array([[right_elbow_yaw]],
np.float32))[0][0]

#             wrist_roll_angle = wrist_roll_angle[0]


            wrist_roll_angle = constrain_angle(wrist_roll_angle, 0, 180)


            left_roll = left_roll + 90

            left_pitch = constrain_angle(left_pitch, -119.5, 119.5)

            left_roll = left_roll + 90

            left_roll = constrain_angle(left_roll, 15, 165)


            left_elbow_roll = constrain_angle(left_elbow_roll, -89.5, -0.5)

            left_elbow_yaw = abs(left_elbow_yaw - 90)

            left_elbow_yaw = constrain_angle(left_elbow_yaw, 0, 180)




            right_pitch = constrain_angle(right_pitch, -119.5, 119.5)


            right_roll = abs(right_roll -90)

            right_roll = constrain_angle(right_roll, 15, 165)


            right_elbow_roll = constrain_angle(right_elbow_roll, 0.5, 89.5)


            right_elbow_yaw = right_elbow_yaw + 90

            right_elbow_yaw = constrain_angle(right_elbow_yaw, 0, 180)


            kalman_left_pitch.predict()

            kalman_left_roll.predict()

            kalman_left_elbow_roll.predict()
```

```
        kalman_left_elbow_yaw.predict()



        kalman_right_pitch.predict()
        kalman_right_roll.predict()

        kalman_right_elbow_roll.predict()
        kalman_right_elbow_yaw.predict()

        kalman_prediction = kalman_wrist_roll.predict()

        kalman_wrist_estimate = kalman_wrist_roll.correct(np.array([[wrist_roll_angle]],
np.float32))

        #wrist_roll_angle = float(wrist_roll_angle[0])

        cv2.putText(frame, f"right_pitch: {right_pitch:.2f}", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.putText(frame, f"right Roll: {right_roll:.2f}", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        #cv2.putText(frame, f'Right Elbow roll: {right_elbow_roll:.2f}', (50, 150),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        #cv2.putText(frame, f'Right wrist roll: {wrist_roll_angle[0][0]:.2f}', (50, 150),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.putText(frame, f'Wrist Roll: {kalman_wrist_estimate[0][0]:.2f} deg', (50,
150),cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 20, 0), 2)
        print("wrist_roll_angle:", wrist_roll_angle)

        cv2.putText(frame, f"Right Elbow Yaw: {right_elbow_yaw:.2f}", (50, 200),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        #cv2.putText(frame, f"Left Pitch: {left_pitch:.2f}", (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        #cv2.putText(frame, f"Left Roll: {left_roll:.2f}", (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
```

```
            #cv2.putText(frame, f"Left Elbow Yaw: {left_elbow_yaw:.2f}", (50, 150),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
            #cv2.putText(frame, f'Left Elbow roll: {left_elbow_roll:.2f}', (50, 200),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            #cv2.putText(frame, f"Head Yaw: {head_yaw:.2f}", (50, 150),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)
            #cv2.putText(frame, f"Head Pitch: {head_pitch:.2f}", (50, 200),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 0), 2)


   left_hand_value = 1.0
   right_hand_value = 1.0

   if results_hands.multi_hand_landmarks and results_hands.multi_handedness:
      for i, hand_landmarks in enumerate(results_hands.multi_hand_landmarks):
         handedness_label = results_hands.multi_handedness[i].classification[0].label

         if handedness_label == 'Right':
            left_hand_value = is_hand_open(hand_landmarks.landmark)
            cv2.putText(frame, f"Left Hand Openness: {left_hand_value:.2f}", (50, 300),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

         elif handedness_label == 'Left':
            right_hand_value = is_hand_open(hand_landmarks.landmark)
            cv2.putText(frame, f"Right Hand Openness: {right_hand_value:.2f}", (50, 350),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


   mp_drawing.draw_landmarks(frame, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
   pose_msg = Float64MultiArray()
   pose_msg.data = [right_roll, right_pitch, right_elbow_yaw, wrist_roll_angle, 0, right_hand_value]
   pub.publish(pose_msg)
   rospy.loginfo(f"Published Angles: {pose_msg.data}")
```

```
    # Display the frame
    cv2.imshow("Pepper Angles", frame)
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```