Documentation Complète du prjet "Follow me Robot"

13 mai 2025

Table des matières

1 Liste des Constantes et Variables

Cette section présente toutes les constantes et variables utilisées dans le code Python pour le système de suivi de personne par un robot, avec leurs descriptions détaillées en français.

1.1 Constantes

FPS = 30
: Fréquence d'images par seconde pour les flux couleur et profondeur de la caméra RealSense, équilibrant vitesse et fluidité.

COLOR_WIDTH = 640
: Largeur (pixels) du flux couleur, pour la visualisation et le calcul du centre de l'image.

COLOR_HEIGHT = 480
: Hauteur (pixels) du flux couleur, pour la mise à l'échelle des coordonnées y.

DEPTH_WIDTH = 640
: Largeur (pixels) du flux de profondeur, pour un mappage précis avec le flux couleur.

DEPTH_HEIGHT = 480
: Hauteur (pixels) du flux de profondeur, correspondant au flux couleur.

MAX_LIN = 0.6
: Vitesse linéaire maximale (m/s) du robot, limitant la vitesse avant (V_x).

MAX_ANGULAR = 0.6
: Vitesse angulaire maximale (rad/s), limitant la rotation (W_z).

TARGET_DISTANCE = 2
: Distance cible (mètres) entre le robot et la personne suivie.

MIN_STANDBY_DISTANCE = 1.90
: Distance minimale (mètres) de la plage d'arrêt où le robot s'immobilise.

MAX_STANDBY_DISTANCE = 2.10
: Distance maximale (mètres) de la plage d'arrêt.

KP_LINEAR = 0.7
: Gain proportionnel pour la vitesse linéaire, ajustant V_x selon l'erreur de distance.

KP_ANGULAR = 1.2
: Gain proportionnel pour la vitesse angulaire, ajustant W_z pour centrer la cible.

SCALE = 100
: Facteur de mise à l'échelle pour convertir les vitesses flottantes en entiers pour Modbus.

LINEAR_SPEED_REGISTER = 100
: Adresse du registre Modbus pour la vitesse linéaire.

ANGULAR_SPEED_REGISTER = 101
: Adresse du registre Modbus pour la vitesse angulaire.

ANALOG_DEAD_ZONE = 0.01
: Seuil minimal de vitesse pour considérer un mouvement.

GESTURE_COOLDOWN_FRAMES = 60
: Nombre d'images ($3\,s$ à $20\,Hz$) avant de permettre un nouveau geste.

GESTURE_CONFIRM_FRAMES = 1
Nombre d'images consécutives pour confirmer un geste.

NO_TARGET_TIMEOUT_FRAMES = 200
Nombre d'images ($10\,s$ à $20\,Hz$) avant réinitialisation si aucune cible.

LOOP_RATE = 0.05
Période cible (secondes) pour la boucle principale ($20\,Hz$).

SMOOTHING_FACTOR = 0.7
Poids pour lisser les coordonnées et vitesses (70% nouvelle donnée, 30% précédente).

SHOULDER_LOSS_HOLD_FRAMES = 200
Nombre d'images ($10\,s$ à $20\,Hz$) pour conserver la dernière position si les épaules sont perdues.

HAND_VISIBILITY_THRESHOLD = 0.1
Confiance minimale pour les points de main dans le geste primaire.

FALLBACK_VISIBILITY_THRESHOLD = 0.1
Confiance minimale pour le poignet/épaule dans le geste de secours.

THUMB_INDEX_DISTANCE_THRESHOLD = 0.07
Distance normalisée minimale pouce-index pour main ouverte.

ALIGNMENT_MARGIN = 50
Différence maximale (pixels) pour l'alignement du bras dans le geste primaire.

1.2 Variables

modbus_client
Objet ModbusClient pour la connexion TCP avec le robot.

pipe
Objet rs.pipeline pour capturer les flux couleur et profondeur.

cfg Objet rs.config pour configurer les flux du pipeline.

align
Objet rs.align pour aligner la profondeur sur le flux couleur.

context
Objet rs.context pour vérifier les appareils RealSense.

profile
Objet rs.pipeline_profile stockant la configuration active.

depth_sensor
Objet rs.sensor pour le capteur de profondeur.

depth_scale
Flottant pour convertir les unités de profondeur en mètres.

Vx Flottant, vitesse linéaire (m/s) envoyée au robot.

Wz Flottant, vitesse angulaire (rad/s) pour la rotation.

pipeline_started
Booléen indiquant si le pipeline est démarré.

target_body_id
Entier ou None, identifie la personne suivie.

is_following
Booléen indiquant si le robot suit la cible.

last_following_state
Booléen stockant l'état précédent de suivi.

gesture_mode
Booléen activant/désactivant la détection de gestes.

gesture_cooldown
Entier comptant les images après un geste.

gesture_confirm_counter
Entier comptant les images pour confirmer un geste.

no_target_frames
Entier comptant les images sans cible.

shoulder_loss_frames
Entier comptant les images sans détection d'épaules.

chest_x_smooth
Flottant ou None, coordonnée x lissée de la poitrine.

chest_y_smooth
Flottant ou None, coordonnée y lissée de la poitrine.

chest_z_smooth
Flottant ou None, profondeur lissée de la poitrine.

Vx_smooth
Flottant, vitesse linéaire lissée.

Wz_smooth
Flottant, vitesse angulaire lissée.

last_valid_chest_x
Flottant ou None, dernière coordonnée x valide.

last_valid_chest_y
Flottant ou None, dernière coordonnée y valide.

last_valid_chest_z
Flottant ou None, dernière profondeur valide.

start_time
Flottant, temps de début d'une itération.

frames
Objet rs.composite_frame contenant les images brutes.

aligned_frames
Objet rs.composite_frame avec images alignées.

color_frame
Objet rs.frame pour l'image couleur.

depth_frame
Objet rs.frame pour l'image de profondeur.

color_image
Tableau Numpy, image couleur (BGR).

depth_image
Tableau Numpy, image de profondeur.

image_rgb
Tableau Numpy, image couleur en RGB.

results
Objet MediaPipe contenant les points de repère.

image_bgr
Tableau Numpy, image RGB reconvertie en BGR.

shoulder_left, shoulder_right
Objets MediaPipe pour les épaules.

x_left, x_right, y_left, y_right
Entiers, coordonnées des épaules.

chest_center_x, chest_center_y, chest_center_z
Flottants, coordonnées actuelles de la poitrine.

right_wrist, right_shoulder, right_elbow
Objets MediaPipe pour le poignet, l'épaule et le coude droits.

x_right_wrist, y_right_wrist
Entiers, coordonnées du poignet droit.

wrist_depth
Flottant, profondeur au poignet.

thumb_tip, index_tip, middle_tip
Objets MediaPipe pour les extrémités des doigts.

thumb_index_distance
Flottant, distance pouce-index.

elbow_x, elbow_y, shoulder_x, shoulder_y, wrist_x, wrist_y
Entiers, coordonnées pour l'alignement du bras.

slope_diff, slope_diff_pixels
Flottants, mesure de l'alignement du bras.

gesture_detected
Booléen indiquant un geste détecté.

image_center_x
Flottant, centre horizontal de l'image.

distance_error
Flottant, erreur de distance à la cible.

angular_error
Flottant, erreur angulaire pour centrer la cible.

is_moving
Booléen indiquant si le robot bouge.

Vx_scaled, Wz_scaled
Entiers, vitesses mises à l'échelle.

Vx_16bit, Wz_16bit
Entiers, vitesses en 16 bits signé.

packed_Vx, packed_Wz
Octets, vitesses packées pour Modbus.

Vx_uint16, Wz_uint16
Entiers, vitesses non signées pour Modbus.

end_time
Flottant, temps de fin d'une itération.

elapsed
Flottant, temps écoulé pour une itération.

sleep_time
Flottant, temps de pause pour maintenir la fréquence.

key Entier, valeur ASCII de la touche pressée.

mode_text, gesture_mode_text, status_text
Chaînes, textes pour l'affichage.

gesture_mode_color
Tuple RVB, couleur pour le texte du mode geste.

device
Objet rs.device pour un appareil RealSense.

sensor
Objet rs.sensor pour un capteur RealSense.

2 Blocs de Code et Explications

Le code est divisé en blocs logiques, chacun avec le code complet et une explication de son rôle.

2.1 Bloc 1 : Importations et Initialisation de MediaPipe

```
import pyrealsense2 as rs
import numpy as np
import cv2
import mediapipe as mp
from pyModbusTCP.client import ModbusClient
import time
import ctypes
import struct

mp_drawing = mp.solutions.drawing_utils
mp_holistic = mp.solutions.holistic
```

— **But** : Importer les bibliothèques pour la capture vidéo (RealSense), le traitement d'image (OpenCV, MediaPipe), la communication (Modbus), et les utilitaires (temps, conversion).

— **Rôle** : Initialiser les utilitaires MediaPipe pour dessiner les points de repère et détecter la pose et les mains.

2.2 Bloc 2 : Définition des Constantes

```
FPS = 30
COLOR_WIDTH = 640
COLOR_HEIGHT = 480
DEPTH_WIDTH = 640
DEPTH_HEIGHT = 480
MAX_LIN = 0.6
MAX_ANGULAR = 0.6
TARGET_DISTANCE = 2
MIN_STANDBY_DISTANCE = 1.90
MAX_STANDBY_DISTANCE = 2.10
KP_LINEAR = 0.7
KP_ANGULAR = 1.2
SCALE = 100
LINEAR_SPEED_REGISTER = 100
ANGULAR_SPEED_REGISTER = 101
ANALOG_DEAD_ZONE = 0.01
GESTURE_COOLDOWN_FRAMES = 60
GESTURE_CONFIRM_FRAMES = 1
NO_TARGET_TIMEOUT_FRAMES = 200
LOOP_RATE = 0.05
SMOOTHING_FACTOR = 0.7
SHOULDER_LOSS_HOLD_FRAMES = 200
HAND_VISIBILITY_THRESHOLD = 0.1
FALLBACK_VISIBILITY_THRESHOLD = 0.1
THUMB_INDEX_DISTANCE_THRESHOLD = 0.07
ALIGNMENT_MARGIN = 50
```

— **But** : Définir les paramètres fixes pour la caméra, le contrôle du robot, la communication Modbus, et la détection de gestes.

— **Rôle** : Configure la résolution, les seuils, les limites de vitesse, et les délais.

2.3 Bloc 3 : Initialisation du Client Modbus

```
modbus_client = ModbusClient(host="127.0.0.1", port=1502)
```

— **But** : Créer un objet ModbusClient pour communiquer avec le robot via TCP à l'adresse 127.0.0.1 :1502.

— **Rôle** : Prépare la connexion pour envoyer les commandes de vitesse.

2.4 Bloc 4 : Fonction de Connexion Modbus

```
def open_modbus_connection():
    if not modbus_client.is_open:
        try:
            if modbus_client.open():
                print("Modbus connection opened successfully.")
                return True
```

```
            else:
                print("Failed to open Modbus connection. Running
                    in Simulation Mode.")
                return False
        except Exception as e:
            print(f"Error opening Modbus connection: {e}")
            return False
    return True
```

— **But** : Tenter d'ouvrir la connexion Modbus et gérer les erreurs.

— **Rôle** : Établit la communication ou passe en mode simulation si la connexion échoue.

2.5 Bloc 5 : Fonction de Calcul de Profondeur Moyenne

```
def get_average_depth(depth_image, x, y, size=5):
    x, y = int(x), int(y)
    half_size = size // 2
    x_min = max(0, x - half_size)
    x_max = min(depth_image.shape[1], x + half_size + 1)
    y_min = max(0, y - half_size)
    y_max = min(depth_image.shape[0], y + half_size + 1)
    region = depth_image[y_min:y_max, x_min:x_max]
    valid_depths = region[region > 0]
    return np.mean(valid_depths) if valid_depths.size > 0 else 0
```

— **But** : Calculer la profondeur moyenne dans une région carrée (5×5 pixels) autour d'un point (x, y).

— **Rôle** : Fournit des mesures précises pour la poitrine et le poignet, en excluant les valeurs invalides.

2.6 Bloc 6 : Initialisation du Pipeline RealSense

```
print("Setting up RealSense pipeline...")
pipe = rs.pipeline()
cfg = rs.config()
cfg.enable_stream(rs.stream.color, COLOR_WIDTH, COLOR_HEIGHT, rs.
    format.bgr8, FPS)
cfg.enable_stream(rs.stream.depth, DEPTH_WIDTH, DEPTH_HEIGHT, rs.
    format.z16, FPS)
align = rs.align(rs.stream.color)
```

— **But** : Configurer le pipeline pour capturer les flux couleur (640x480, BGR, 30 FPS) et profondeur (640x480, 16 bits, 30 FPS).

— **Rôle** : Prépare la caméra pour fournir des images alignées.

2.7 Bloc 7 : Vérification des Appareils RealSense

```
context = rs.context()
if not context.devices:
    print("Error: No RealSense device connected!")
    open_modbus_connection()
    try:
        modbus_client.write_multiple_registers(
           LINEAR_SPEED_REGISTER, [0, 0])
        print("Sent initial stop command to Modbus.")
    except Exception as e:
        print(f"Failed to send initial stop command: {e}")
    exit(1)
else:
    print(f"Found {len(context.devices)} RealSense device(s):")
    for device in context.devices:
        print(f" - {device.get_info(rs.camera_info.name)} (Serial
           : {device.get_info(rs.camera_info.serial_number)})")
```

— **But** : Vérifier la présence d'une caméra RealSense et arrêter si aucune n'est trouvée.

— **Rôle** : Affiche les appareils détectés et tente une commande d'arrêt via Modbus.

2.8 Bloc 8 : Test de Connexion Modbus

```
if open_modbus_connection():
    try:
        modbus_client.write_multiple_registers(
           LINEAR_SPEED_REGISTER, [0, 0])
        print("Successfully sent test stop command to Modbus.")
    except Exception as e:
        print(f"Failed to send test stop command: {e}")
```

— **But** : Tester la connexion Modbus en envoyant une commande d'arrêt.

— **Rôle** : Vérifie la communication avec le robot avant de commencer.

2.9 Bloc 9 : Démarrage du Pipeline et Initialisation de MediaPipe

```
try:
    print("Starting pipeline...")
    profile = pipe.start(cfg)
    print("Pipeline started!")
    depth_sensor = profile.get_device().first_depth_sensor()
    depth_scale = depth_sensor.get_depth_scale()
    with mp_holistic.Holistic(
        min_detection_confidence=0.5,
        min_tracking_confidence=0.5,
        enable_segmentation=False,
        refine_face_landmarks=False,
```

```
        model_complexity=1
    ) as holistic:
```

— **But** : Démarrer le pipeline RealSense et initialiser le modèle Holistic de MediaPipe.
— **Rôle** : Obtient l'échelle de profondeur et prépare la détection des points de repère.

2.10 Bloc 10 : Initialisation des Variables

```
Vx = 0.0
Wz = 0.0
pipeline_started = True
target_body_id = None
is_following = False
last_following_state = False
gesture_mode = False
gesture_cooldown = 0
gesture_confirm_counter = 0
no_target_frames = 0
shoulder_loss_frames = 0
chest_x_smooth = None
chest_y_smooth = None
chest_z_smooth = None
Vx_smooth = 0.0
Wz_smooth = 0.0
last_valid_chest_x = None
last_valid_chest_y = None
last_valid_chest_z = None
```

— **But** : Initialiser les variables pour les vitesses, l'état du suivi, les gestes, et les coordonnées.
— **Rôle** : Fournit des valeurs de départ pour éviter des erreurs.

2.11 Bloc 11 : Boucle Principale

```
while True:
    try:
        start_time = time.time()
        frames = pipe.wait_for_frames()
        aligned_frames = align.process(frames)
        color_frame = aligned_frames.get_color_frame()
        depth_frame = aligned_frames.get_depth_frame()
        if not color_frame or not depth_frame:
            print("No frame received, skipping...")
            Vx = 0.0
            Wz = 0.0
            no_target_frames += 1
            continue
```

— **But** : Capturer et traiter les images couleur et profondeur dans une boucle infinie.
— **Rôle** : Orchestre la capture, saute les itérations sans images, et met à jour l'état.

2.12 Bloc 12 : Traitement des Images et Détection MediaPipe

```
color_image = np.asanyarray(color_frame.get_data())
depth_image = np.asanyarray(depth_frame.get_data())
image_rgb = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)
results = holistic.process(image_rgb)
image_bgr = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
```

— **But** : Convertir les images en tableaux Numpy, passer en RGB pour MediaPipe, détecter les points de repère, et revenir en BGR.

— **Rôle** : Prépare les données pour le suivi et la visualisation.

2.13 Bloc 13 : Dessin des Points de Repère

```
if results.pose_landmarks:
    mp_drawing.draw_landmarks(
        image_bgr,
        results.pose_landmarks,
        mp_holistic.POSE_CONNECTIONS,
        mp_drawing.DrawingSpec(color=(245,117,66), thickness=2,
           circle_radius=4),
        mp_drawing.DrawingSpec(color=(245,66,230), thickness=2,
           circle_radius=2)
    )
    if results.right_hand_landmarks:
        mp_drawing.draw_landmarks(
            image_bgr,
            results.right_hand_landmarks,
            mp_holistic.HAND_CONNECTIONS,
            mp_drawing.DrawingSpec(color=(0,0,255), thickness=2,
               circle_radius=2),
            mp_drawing.DrawingSpec(color=(0,255,255), thickness
               =2, circle_radius=2)
        )
```

— **But** : Dessiner les points de repère de la pose et de la main droite sur l'image.

— **Rôle** : Visualise les détections pour le suivi et les gestes.

2.14 Bloc 14 : Verrouillage de la Cible

```
if target_body_id is None:
    target_body_id = 1
    is_following = False
    last_following_state = False
    print(f"Locked Target (Body ID {target_body_id}), Restored
       mode: {'Following' if is_following else 'Not Following'}")
```

— **But** : Verrouiller la première personne détectée comme cible.

— **Rôle** : Initialise le suivi avec un identifiant fixe (1).

2.15 Bloc 15 : Calcul des Coordonnées de la Poitrine

```
if target_body_id is not None:
    shoulder_left = results.pose_landmarks.landmark[11]
    shoulder_right = results.pose_landmarks.landmark[12]
    if shoulder_left.visibility > 0.3 and shoulder_right.
       visibility > 0.3:
        x_left = int(shoulder_left.x * COLOR_WIDTH)
        x_right = int(shoulder_right.x * COLOR_WIDTH)
        y_left = int(shoulder_left.y * COLOR_HEIGHT)
        y_right = int(shoulder_right.y * COLOR_HEIGHT)
        if (0 <= x_left < DEPTH_WIDTH and 0 <= y_left <
           DEPTH_HEIGHT and
            0 <= x_right < DEPTH_WIDTH and 0 <= y_right <
               DEPTH_HEIGHT):
            chest_center_x = (x_left + x_right) / 2
            chest_center_y = (y_left + y_right) / 2
            chest_center_z = get_average_depth(depth_image,
               chest_center_x, chest_center_y) * depth_scale
```

— **But** : Calculer les coordonnées du centre de la poitrine à partir des épaules.

— **Rôle** : Convertit les coordonnées normalisées en pixels et obtient la profondeur.

2.16 Bloc 16 : Lissage des Coordonnées et Visualisation

```
if 0.5 < chest_center_z < 5.0:
    if chest_x_smooth is None:
        chest_x_smooth = chest_center_x
        chest_y_smooth = chest_center_y
        chest_z_smooth = chest_center_z
    else:
        chest_x_smooth = SMOOTHING_FACTOR * chest_center_x + (1 -
            SMOOTHING_FACTOR) * chest_x_smooth
        chest_y_smooth = SMOOTHING_FACTOR * chest_center_y + (1 -
            SMOOTHING_FACTOR) * chest_y_smooth
        chest_z_smooth = SMOOTHING_FACTOR * chest_center_z + (1 -
            SMOOTHING_FACTOR) * chest_z_smooth
    last_valid_chest_x = chest_x_smooth
    last_valid_chest_y = chest_y_smooth
    last_valid_chest_z = chest_z_smooth
    cv2.circle(image_bgr, (int(chest_x_smooth), int(
       chest_y_smooth)), 5, (0, 0, 255), -1)
    cv2.putText(image_bgr, f"Distance: {chest_z_smooth:.2f} m", (
       int(chest_x_smooth), int(chest_y_smooth - 10)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 0, 255), 1)
    no_target_frames = 0
    shoulder_loss_frames = 0
```

— **But** : Lisser les coordonnées de la poitrine et afficher un marqueur avec la distance.

— **Rôle** : Réduit les secousses, met à jour les positions valides, et réinitialise les compteurs.

2.17 Bloc 17 : Détection de Gestes

```
if gesture_mode and gesture_cooldown <= 0:
    right_wrist = results.pose_landmarks.landmark[16]
    right_shoulder = results.pose_landmarks.landmark[12]
    right_elbow = results.pose_landmarks.landmark[14]
    x_right_wrist = int(right_wrist.x * COLOR_WIDTH)
    y_right_wrist = int(right_wrist.y * COLOR_HEIGHT)
    wrist_depth = get_average_depth(depth_image, x_right_wrist,
       y_right_wrist) * depth_scale
    gesture_detected = False
    print(f"Hand landmarks detected: {bool(results.
       right_hand_landmarks)}")
    if (right_wrist.visibility > HAND_VISIBILITY_THRESHOLD and
        right_shoulder.visibility > HAND_VISIBILITY_THRESHOLD and
        right_elbow.visibility > HAND_VISIBILITY_THRESHOLD and
        results.right_hand_landmarks and
        0.5 <= wrist_depth <= 5.0):
        thumb_tip = results.right_hand_landmarks.landmark[4]
        index_tip = results.right_hand_landmarks.landmark[8]
        middle_tip = results.right_hand_landmarks.landmark[12]
        if (thumb_tip.visibility > HAND_VISIBILITY_THRESHOLD and
            index_tip.visibility > HAND_VISIBILITY_THRESHOLD and
            middle_tip.visibility > HAND_VISIBILITY_THRESHOLD):
            thumb_index_distance = np.sqrt(
                (thumb_tip.x - index_tip.x)**2 +
                (thumb_tip.y - index_tip.y)**2
            )
            elbow_x = int(right_elbow.x * COLOR_WIDTH)
            elbow_y = int(right_elbow.y * COLOR_HEIGHT)
            shoulder_x = int(right_shoulder.x * COLOR_WIDTH)
            shoulder_y = int(right_shoulder.y * COLOR_HEIGHT)
            wrist_x = x_right_wrist
            wrist_y = y_right_wrist
            slope_diff = abs((shoulder_y - elbow_y) / (shoulder_x
                - elbow_x + 1e-6) -
                             (wrist_y - shoulder_y) / (wrist_x -
                                shoulder_x + 1e-6))
            slope_diff_pixels = abs((shoulder_y - elbow_y) - (
               wrist_y - shoulder_y))
            print(f"Wrist y={right_wrist.y:.2f}, Shoulder y={
               right_shoulder.y:.2f}, "
                f"Thumb-index distance={thumb_index_distance:.2
                   f}, "
                f"Visibility: Wrist={right_wrist.visibility:.2f
                   }, "
                f"Thumb={thumb_tip.visibility:.2f}, Index={
                   index_tip.visibility:.2f}, "
                f"Middle={middle_tip.visibility:.2f}, "
                f"Slope diff pixels={slope_diff_pixels:.2f},
                   Depth={wrist_depth:.2f}, "
```

```
                    f"Coords: ({x_right_wrist}, {y_right_wrist})")
            if (0 <= x_right_wrist < DEPTH_WIDTH and 0 <=
               y_right_wrist < DEPTH_HEIGHT and
                right_wrist.y < right_shoulder.y and
                thumb_index_distance >
                   THUMB_INDEX_DISTANCE_THRESHOLD and
                slope_diff_pixels < ALIGNMENT_MARGIN):
                gesture_confirm_counter += 1
                print(f"Primary gesture detected, confirming: {
                   gesture_confirm_counter}/{
                   GESTURE_CONFIRM_FRAMES}")
                radius = 10 + (gesture_confirm_counter * 2)
                cv2.circle(image_bgr, (x_right_wrist,
                   y_right_wrist), radius, (0, 255, 0), 2)
                cv2.line(image_bgr, (elbow_x, elbow_y), (
                   shoulder_x, shoulder_y), (0, 255, 0), 2)
                cv2.line(image_bgr, (shoulder_x, shoulder_y), (
                   wrist_x, wrist_y), (0, 255, 0), 2)
                if gesture_confirm_counter >=
                   GESTURE_CONFIRM_FRAMES:
                    gesture_detected = True
                    print(f"Primary gesture confirmed: {'
                       Following' if not is_following else 'Not
                       Following'} activated")
            else:
                gesture_confirm_counter = 0
                print("Primary gesture conditions not met:
                   Insufficient y-distance, thumb-index distance,
                    or alignment")
                cv2.circle(image_bgr, (x_right_wrist,
                   y_right_wrist), 10, (0, 255, 255), 2)
        else:
            gesture_confirm_counter = 0
            print("Hand landmark visibility too low")
            cv2.circle(image_bgr, (x_right_wrist, y_right_wrist),
                10, (0, 255, 255), 2)
    else:
        gesture_confirm_counter = 0
        print("Primary gesture failed: Low visibility or out of
           depth range")
        cv2.circle(image_bgr, (x_right_wrist, y_right_wrist), 10,
            (0, 255, 255), 2)
    if not gesture_detected and right_wrist.visibility >
       FALLBACK_VISIBILITY_THRESHOLD and right_shoulder.
       visibility > HAND_VISIBILITY_THRESHOLD:
        print(f"Fallback check: Wrist y={right_wrist.y:.2f},
           Shoulder y={right_shoulder.y:.2f}, "
              f"Visibility: Wrist={right_wrist.visibility:.2f},
                 Shoulder={right_shoulder.visibility:.2f}, "
              f"Depth={wrist_depth:.2f}, Coords: ({x_right_wrist
                 }, {y_right_wrist})")
```

```
        if (0 <= x_right_wrist < DEPTH_WIDTH and 0 <=
           y_right_wrist < DEPTH_HEIGHT and
            right_wrist.y < right_shoulder.y and
            0.5 <= wrist_depth <= 5.0):
            gesture_confirm_counter += 1
            print(f"Fallback gesture detected, confirming: {
               gesture_confirm_counter}/{GESTURE_CONFIRM_FRAMES
               }")
            radius = 10 + (gesture_confirm_counter * 2)
            cv2.circle(image_bgr, (x_right_wrist, y_right_wrist),
                radius, (0, 255, 128), 2)
            if gesture_confirm_counter >= GESTURE_CONFIRM_FRAMES:
                gesture_detected = True
                print(f"Fallback gesture confirmed: {'Following'
                   if not is_following else 'Not Following'}
                   activated")
        else:
            gesture_confirm_counter = 0
            print("Fallback gesture conditions not met:
               Insufficient y-distance or out of depth range")
            cv2.circle(image_bgr, (x_right_wrist, y_right_wrist),
                10, (0, 255, 255), 2)
    if gesture_detected:
        is_following = not is_following
        gesture_cooldown = GESTURE_COOLDOWN_FRAMES
        gesture_confirm_counter = 0
if gesture_cooldown > 0:
    gesture_cooldown -= 1
```

— **But** : Détecter les gestes primaires (main ouverte, bras aligné) ou de secours (poignet levé) pour basculer le mode de suivi.

— **Rôle** : Vérifie la visibilité, la profondeur, et l'alignement, affiche des marqueurs visuels, et impose un délai après un geste.

2.18 Bloc 18 : Calcul des Vitesses

```
Vx = 0.0
Wz = 0.0
if is_following:
    image_center_x = COLOR_WIDTH / 2
    if MIN_STANDBY_DISTANCE <= chest_z_smooth <=
       MAX_STANDBY_DISTANCE:
        Vx = 0.0
        Wz = 0.0
        Vx_smooth = 0.0
        Wz_smooth = 0.0
        print(f"In standby range: Distance={chest_z_smooth:.2f} m
           , Vx_smooth={Vx_smooth:.2f}, Wz_smooth={Wz_smooth:.2f
           }")
    else:
```

```
        distance_error = chest_z_smooth - TARGET_DISTANCE
        Vx = KP_LINEAR * distance_error
        Vx = max(0.0, Vx)
        Vx = np.clip(Vx, 0.0, MAX_LIN)
        angular_error = (image_center_x - chest_x_smooth) /
           image_center_x
        Wz = KP_ANGULAR * angular_error
        Wz = np.clip(Wz, -MAX_ANGULAR, MAX_ANGULAR)
        print(f"Following: Distance={chest_z_smooth:.2f} m, Vx={
           Vx:.2f}, Wz={Wz:.2f}")
Vx_smooth = SMOOTHING_FACTOR * Vx + (1 - SMOOTHING_FACTOR) *
   Vx_smooth
Wz_smooth = SMOOTHING_FACTOR * Wz + (1 - SMOOTHING_FACTOR) *
   Wz_smooth
Vx = Vx_smooth
Wz = Vz_smooth
```

— **But** : Calculer les vitesses linéaire et angulaire pour maintenir la distance cible et centrer la personne.

— **Rôle** : Utilise un contrôle proportionnel, arrête le robot dans la plage d'arrêt, et lisse les vitesses.

2.19 Bloc 19 : Gestion de la Perte des Épaules

```
else:
    print("Target out of depth range, holding last mode...")
    shoulder_loss_frames += 1
else:
    print("Target out of frame, holding last mode...")
    shoulder_loss_frames += 1
else:
    print("Shoulders not detected, holding last mode...")
    shoulder_loss_frames += 1
```

— **But** : Gérer les cas où la cible est hors de portée, hors cadre, ou les épaules non détectées.

— **Rôle** : Incrémente le compteur de perte pour déclencher des actions ultérieures.

2.20 Bloc 20 : Gestion du Mode Attente

```
if shoulder_loss_frames > 0 and shoulder_loss_frames <=
   SHOULDER_LOSS_HOLD_FRAMES:
    if last_valid_chest_x is not None:
        chest_x_smooth = last_valid_chest_x
        chest_y_smooth = last_valid_chest_y
        chest_z_smooth = last_valid_chest_z
        cv2.circle(image_bgr, (int(chest_x_smooth), int(
           chest_y_smooth)), 5, (0, 255, 255), -1)
```

```
        cv2.putText(image_bgr, f"Distance: {chest_z_smooth:.2f} m (Hold)", (int(chest_x_smooth), int(chest_y_smooth - 10)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 255), 1)
        print(f"Holding last mode: {'Following' if is_following else 'Not Following'}, Frames: {shoulder_loss_frames}/{SHOULDER_LOSS_HOLD_FRAMES}")
        no_target_frames = 0
    else:
        Vx = 0.0
        Wz = 0.0
        no_target_frames += 1
elif shoulder_loss_frames > SHOULDER_LOSS_HOLD_FRAMES:
    print("Shoulder loss timeout, stopping robot...")
    Vx = 0.0
    Wz = 0.0
    no_target_frames += 1
    shoulder_loss_frames = 0
    last_valid_chest_x = None
    last_valid_chest_y = None
    last_valid_chest_z = None
```

— **But** : Conserver les dernières coordonnées valides pendant une perte temporaire ou arrêter le robot après un délai.

— **Rôle** : Affiche un marqueur jaune en mode attente et réinitialise si la perte persiste.

2.21 Bloc 21 : Gestion de la Perte de Cible

```
else:
    print("No target detected, stopping robot...")
    Vx = 0.0
    Wz = 0.0
    no_target_frames += 1
else:
    print("No pose landmarks detected, stopping robot...")
    Vx = 0.0
    Wz = 0.0
    no_target_frames += 1
    if no_target_frames >= NO_TARGET_TIMEOUT_FRAMES:
        print("Target lost for too long, resetting tracking...")
        last_following_state = is_following
        is_following = False
        target_body_id = None
        no_target_frames = 0
        shoulder_loss_frames = 0
        last_valid_chest_x = None
        last_valid_chest_y = None
        last_valid_chest_z = None
```

```
21          cv2.putText(image_bgr, "Target Lost - Robot Stopped",
               (10, 65), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255),
               2)
```

— **But** : Arrêter le robot si aucune cible ou pose n'est détectée, et réinitialiser après un délai.

— **Rôle** : Gère la perte complète de la cible et affiche un message de statut.

2.22 Bloc 22 : Affichage du Statut

```
1 mode_text = "Real Mode" if modbus_client.is_open else "Simulation
      Mode"
2 cv2.putText(image_bgr, mode_text, (10, 20), cv2.
     FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
3 gesture_mode_text = "Gesture Mode: ON (Press 'u' to disable)" if
     gesture_mode else "Gesture Mode: OFF (Press 'c' to enable)"
4 gesture_mode_color = (0, 255, 0) if gesture_mode else (255, 255,
     255)
5 cv2.putText(image_bgr, gesture_mode_text, (10, 35), cv2.
     FONT_HERSHEY_SIMPLEX, 0.4, (gesture_mode_color), 2)
6 status_text = "Following" if is_following else "Not Following"
7 cv2.putText(image_bgr, status_text, (10, 50), cv2.
     FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 0) if is_following else
     (0, 0, 255), 2)
```

— **But** : Afficher l'état du mode (réel/simulation), des gestes, et du suivi sur l'image.

— **Rôle** : Fournit un retour visuel sur l'état du système.

2.23 Bloc 23 : Affichage de l'Image

```
1 cv2.imshow('RealSense Color Feed with Landmarks', image_bgr)
```

— **But** : Afficher l'image couleur avec les points de repère et les annotations.

— **Rôle** : Permet une visualisation en temps réel des détections et des statuts.

2.24 Bloc 24 : Gestion des Entrées Clavier

```
1  key = cv2.waitKey(1)
2  if key & 0xFF == ord('q'):
3      print("Quitting...")
4      break
5  elif key == ord('c'):
6      if not gesture_mode:
7          gesture_mode = True
8          print("Gesture mode activated (press 'u' to deactivate)")
9  elif key == ord('u'):
10     if gesture_mode:
11         gesture_mode = False
```

```
        print("Gesture mode deactivated (press 'c' to activate)")
elif key == ord('f'):
    is_following = not is_following
    print(f"Manual toggle: {'Following' if is_following else 'Not
        Following'} activated")
```

— **But** : Gérer les entrées clavier pour quitter ('q'), activer ('c') ou désactiver ('u') le mode geste, ou basculer manuellement le suivi ('f').

— **Rôle** : Permet un contrôle interactif du système.

2.25 Bloc 25 : Communication Modbus

```
try:
    is_moving = abs(Vx) > ANALOG_DEAD_ZONE or abs(Wz) >
       ANALOG_DEAD_ZONE
    if not modbus_client.is_open:
        open_modbus_connection()
    if modbus_client.is_open:
        Vx_scaled = int(Vx * SCALE)
        Wz_scaled = int(Wz * SCALE)
        Vx_16bit = ctypes.c_int16(Vx_scaled).value
        Wz_16bit = ctypes.c_int16(Wz_scaled).value
        packed_Vx = struct.pack('>h', Vx_16bit)
        packed_Wz = struct.pack('>h', Wz_16bit)
        Vx_uint16 = struct.unpack('>H', packed_Vx)[0]
        Wz_uint16 = struct.unpack('>H', packed_Wz)[0]
        print(f"Sending to Modbus: Vx_scaled={Vx_scaled},
           Wz_scaled={Wz_scaled}, Registers=[{Vx_uint16}, {
           Wz_uint16}]")
        modbus_client.write_multiple_registers(
           LINEAR_SPEED_REGISTER, [Vx_uint16, Wz_uint16])
        print(f"Real Mode | Vx: {Vx:.2f} m/s, Wz: {Wz:.2f} rad/s,
            Moving: {is_moving}")
    else:
        print(f"Simulation Mode | Vx: {Vx:.2f} m/s, Wz: {Vz:.2f}
           rad/s, Moving: {is_moving}")
except Exception as e:
    print(f"Error during Modbus communication: {e}")
    print(f"Simulation Mode | Vx: {Vx:.2f} m/s, Wz: {Wz:.2f} rad/
       s, Moving: {is_moving}")
```

— **But** : Envoyer les vitesses au robot via Modbus ou simuler en cas d'échec.

— **Rôle** : Convertit les vitesses en entiers 16 bits non signés et les envoie aux registres 100 et 101.

2.26 Bloc 26 : Contrôle de la Fréquence de Boucle

```
end_time = time.time()
elapsed = end_time - start_time
```

```
print(f"Frame processing time: {elapsed:.3f} s")
sleep_time = max(0, LOOP_RATE - elapsed)
time.sleep(sleep_time)
```

— **But** : Maintenir une fréquence de boucle de $20\,Hz$.

— **Rôle** : Calcule le temps écoulé et ajoute une pause si nécessaire.

2.27 Bloc 27 : Gestion des Erreurs de la Boucle

```
except Exception as e:
    print(f"Error processing frame: {e}")
    continue
```

— **But** : Gérer les erreurs dans la boucle principale.

— **Rôle** : Affiche l'erreur et passe à l'itération suivante.

2.28 Bloc 28 : Gestion des Erreurs du Pipeline

```
except Exception as e:
    print(f"Pipeline error: {e}")
    try:
        device = context.devices[0]
        sensor = device.first_depth_sensor()
        print("Supported depth stream configurations:")
        for stream in sensor.get_stream_profiles():
            if stream.stream_type() == rs.stream.depth:
                print(f" - {stream}")
        sensor = device.query_sensors()[1]
        print("Supported color stream configurations:")
        for stream in sensor.get_stream_profiles():
            if stream.stream_type() == rs.stream.color:
                print(f" - {stream}")
    except Exception as e:
        print(f"Error querying stream configurations: {e}")
```

— **But** : Gérer les erreurs critiques du pipeline et afficher les configurations des flux.

— **Rôle** : Fournit des informations de débogage pour résoudre les problèmes matériels.

2.29 Bloc 29 : Nettoyage Final

```
finally:
    print("Stopping pipeline...")
    if modbus_client.is_open:
        try:
            modbus_client.write_multiple_registers(
               LINEAR_SPEED_REGISTER, [0, 0])
            print("Sent final stop command to Modbus.")
        except Exception as e:
```

```
8                 print(f"Error sending stop command: {e}")
9             modbus_client.close()
10        if pipeline_started:
11            pipe.stop()
12        cv2.destroyAllWindows()
```

— **But** : Arrêter le robot, fermer la connexion Modbus, arrêter le pipeline, et fermer les fenêtres.

— **Rôle** : Assure une terminaison propre du programme.

3 Pseudo-code

Cette section présente deux niveaux de description du fonctionnement du système : un pseudo-code généralisé, accessible à tous, et un pseudo-code détaillé, fidèle au code Python.

3.1 Pseudo-code Généralisé

Le pseudo-code suivant décrit le fonctionnement global du système de manière simple, pour une personne sans connaissances en programmation.

— DÉMARRER le système en préparant la caméra et le robot
— VÉRIFIER que la caméra et le robot sont connectés
— SI la caméra ou le robot n'est pas détecté ALORS
  — AFFICHER un message d'erreur et arrêter le système
— CONFIGURER la caméra pour capturer des images et des distances
— TANT QUE le système fonctionne :
  — CAPTURER une image avec la caméra
  — ANALYSER l'image pour trouver une personne
  — SI une personne est détectée ALORS
    — CHOISIR cette personne comme cible à suivre
    — MESURER la distance entre le robot et la personne
    — DÉTERMINER la position de la personne (au centre, à gauche, à droite)
    — SI le mode de suivi est activé ALORS
      — SI la personne est trop loin ou trop près ALORS
      — DÉPLACER le robot vers la personne
      — SI la personne n'est pas au centre ALORS
      — TOURNER le robot pour centrer la personne
      — SI la personne est à la bonne distance ALORS
      — ARRÊTER le robot
    — VÉRIFIER si la personne fait un geste avec la main
    — SI un geste est détecté ALORS
      — ACTIVER ou DÉSACTIVER le mode de suivi
      — ATTENDRE quelques secondes avant de détecter un nouveau geste

        — AFFICHER l'image avec la position de la personne et l'état du robot
    — SINON
        — ARRÊTER le robot
        — SI aucune personne n'est détectée pendant longtemps ALORS
            — OUBLIER la cible et attendre une nouvelle personne
    — VÉRIFIER si l'utilisateur appuie sur une touche
    — SI la touche "quitter" est pressée ALORS
        — ARRÊTER le système
    — SI une touche spéciale est pressée ALORS
        — ACTIVER ou DÉSACTIVER le mode de suivi ou la détection de gestes
    — ENVOYER les commandes de mouvement au robot
    — ATTENDRE un court instant pour maintenir une vitesse régulière
— À LA FIN
    — ARRÊTER le robot
    — ÉTEINDRE la caméra
    — FERMER l'affichage

3.2 Pseudo-code Détaillé

Le pseudo-code suivant décrit le fonctionnement global du système de suivi de personne, dans un style structuré et fidèle au code Python.

— INITIALISER la caméra RealSense avec des flux couleur (640×480, 30 FPS) et profondeur (640×480, 30 FPS)
— INITIALISER MediaPipe Holistic avec une confiance de 0.5
— INITIALISER le client Modbus à 127.0.0.1 :1502
— DÉFINIR les constantes : FPS = 30, COLOR_WIDTH = 640, TARGET_DISTANCE = 2, etc.
— INITIALISER les variables : Vx = 0.0, Wz = 0.0, target_body_id = None, etc.
— VÉRIFIER les appareils RealSense
— SI aucun appareil RealSense ALORS
    — AFFICHER une erreur, tenter une commande d'arrêt Modbus, et quitter
— TENTER d'ouvrir la connexion Modbus
— SI connexion réussie ALORS
    — ENVOYER une commande d'arrêt pour tester
— SINON
    — PASSER en mode simulation
— DÉMARRER le pipeline RealSense
— OBTENIR l'échelle de profondeur
— TANT QUE le programme est en cours :
    — ENREGISTRER le temps de début

- CAPTURER les images couleur et profondeur
- ALIGNER la profondeur sur le flux couleur
- SI aucune image reçue ALORS
  - DÉFINIR Vx = 0.0, Wz = 0.0
  - INCRÉMENTER no_target_frames
  - CONTINUER
- CONVERTIR les images en tableaux Numpy
- CONVERTIR l'image couleur en RGB pour MediaPipe
- DÉTECTER les points de repère (pose, main droite)
- CONVERTIR l'image RGB en BGR
- SI des points de pose détectés ALORS
  - DESSINER les points de pose et connexions
  - SI main droite détectée ALORS
    - DESSINER les points de main et connexions
  - SI target_body_id est None ALORS
    - DÉFINIR target_body_id = 1, is_following = False
    - AFFICHER "Cible verrouillée"
  - SI target_body_id n'est pas None ALORS
    - OBTENIR les coordonnées des épaules
    - SI visibilité des épaules > 0.3 ET coordonnées dans les limites ALORS
    - CALCULER chest_center_x, y, z
    - SI $0.5 < chest\_center\_z < 5.0$ ALORS
    - LISSER chest_x_smooth, y_smooth, z_smooth
    - METTRE À JOUR last_valid_chest_x, y, z
    - DESSINER un cercle rouge et afficher la distance
    - RÉINITIALISER no_target_frames, shoulder_loss_frames
    - SI gesture_mode activé ET gesture_cooldown = 0 ALORS
    - OBTENIR poignet, épaule, coude droits
    - CALCULER x_right_wrist, y_right_wrist, wrist_depth
    - SI main droite détectée ET visibilité suffisante ALORS
    - OBTENIR pouce, index, majeur
    - CALCULER thumb_index_distance
    - CALCULER l'alignement du bras
    - SI conditions de geste primaire remplies ALORS
    - INCRÉMENTER gesture_confirm_counter
    - DESSINER des marqueurs verts
    - SI confirmé ALORS
    - BASCULER is_following
    - DÉFINIR gesture_cooldown
    - SINON
    - RÉINITIALISER gesture_confirm_counter

- DESSINER un marqueur jaune
- SINON SI conditions de geste de secours remplies ALORS
- INCRÉMENTER gesture_confirm_counter
- DESSINER un marqueur vert clair
- SI confirmé ALORS
- BASCULER is_following
- DÉFINIR gesture_cooldown
- SINON
- RÉINITIALISER gesture_confirm_counter
- DESSINER un marqueur jaune
- SI is_following ALORS
- SI dans la plage d'arrêt ALORS
- DÉFINIR Vx = 0.0, Wz = 0.0
- RÉINITIALISER Vx_smooth, Wz_smooth
- SINON
- CALCULER distance_error
- CALCULER Vx = KP_LINEAR * distance_error
- LIMITER Vx à $[0.0, MAX\_LIN]$
- CALCULER angular_error
- CALCULER Wz = KP_ANGULAR * angular_error
- LIMITER Wz à $[-MAX\_ANGULAR, MAX\_ANGULAR]$
- LISSER Vx_smooth, Wz_smooth
- SINON
- INCRÉMENTER shoulder_loss_frames
- SINON
- INCRÉMENTER shoulder_loss_frames
- SI shoulder_loss_frames > 0 ET $\leq$ SHOULDER_LOSS_HOLD_FRAMES ALORS
- UTILISER les dernières coordonnées valides
- DESSINER un cercle jaune et afficher la distance
- RÉINITIALISER no_target_frames
- SINON SI shoulder_loss_frames $> SHOULDER\_LOSS\_HOLD\_FRAMESALO$
- DÉFINIR Vx = 0.0, Wz = 0.0
- RÉINITIALISER les coordonnées valides
- INCRÉMENTER no_target_frames
- SINON
- DÉFINIR Vx = 0.0, Wz = 0.0
- INCRÉMENTER no_target_frames
- SI no_target_frames $\geq$ NO_TARGET_TIMEOUT_FRAMES ALORS
- RÉINITIALISER target_body_id, is_following
- RÉINITIALISER coordonnées et compteurs

— AFFICHER "Cible perdue"

— AFFICHER mode, gesture_mode, et statut sur l'image

— AFFICHER l'image avec les points de repère

— LIRE la touche pressée

— SI touche = 'q' ALORS

— SORTIR de la boucle

— SI touche = 'c' ALORS

— ACTIVER gesture_mode

— SI touche = 'u' ALORS

— DÉSACTIVER gesture_mode

— SI touche = 'f' ALORS

— BASCULER is_following

— SI Modbus connecté ALORS

— CONVERTIR Vx, Wz en entiers 16 bits non signés

— ENVOYER aux registres 100, 101

— SINON

— AFFICHER les vitesses en mode simulation

— CALCULER le temps écoulé

— ATTENDRE pour maintenir 20 Hz

EN CAS D'ERREUR

— AFFICHER l'erreur et continuer

À LA FIN

— ARRÊTER le robot

— FERMER Modbus

— ARRÊTER le pipeline

— FERMER les fenêtres