Faceto Cristian

5C ROB

VisionHand Control:

Controllo del Manipolatore

tramite Webcam

INDICE

INTRODUZIONE

Benvenuti nel Futuro dell'Interfaccia Uomo-Macchina: VisionHand Control
Nel mondo dell'automazione e della robotica, l'interazione uomo-macchina ha tradizionalmente richiesto l'uso di dispositivi di controllo esterni come joystick, tastiere o mouse. Tuttavia, con l'avvento dell'intelligenza artificiale e dei sensori di visione, stiamo assistendo a un'evoluzione significativa in questo campo.
VisionHand Control rappresenta un passo avanti in questa direzione, sfruttando l'intelligenza artificiale per rilevare e tracciare i movimenti delle mani attraverso l'uso della webcam del PC. Questi movimenti vengono quindi utilizzati per controllare un braccio antropomorfo (detto manipolatore), consentendo un'interazione intuitiva e diretta con la macchina.

Il funzionamento del sistema è abbastanza semplice: la webcam cattura le immagini delle mani dell'utente, esse vengono poi analizzate da un algoritmo di visione artificiale per rilevare e tracciare i movimenti. Queste informazioni vengono quindi elaborate dal programma Python, che converte i gesti e la posizione delle mani in comandi per il manipolatore.

Il manipolatore è composto da quattro servomotori, ognuno dei quali controlla un'articolazione del braccio robotico. La movimentazione sull’asse delle ascisse (x) è controllata dal movimento orizzontale della mano, l’asse delle ordinate (y) è pilotata dalla profondità della mano, l'asse z dal movimento verticale e infine l’apertura e chiusura della pinza è gestita dall’apertura o chiusura a pugno della mano. Combinando questi semplici movimenti rilevati dalla webcam e algoritmi di controllo del manipolatore implementati nel codice Python, è possibile impartire al manipolatore una vasta gamma di comandi, consentendo di eseguire operazioni complesse in modo fluido e preciso.

Il mio obiettivo con VisionHand Control è quello di dimostrare le potenzialità dell'integrazione tra intelligenza artificiale e macchinari robotici nel campo dell'automazione. Vorrei mostrare come l'utilizzo della visione artificiale e dei gesti delle mani, ma anche di tutto il corpo, possa rendere l'interazione uomo-macchina più naturale ed intuitiva, aprendo la strada a nuove applicazioni e scenari d'uso nel mondo reale.

L’ispirazione per questo progetto arriva dal film “Real Steel” dove il robot pugile (Atom) nell’ultimo incontro viene controllato in modalità ombra, cioè in cui il robot segue a vista le mosse di chi lo controlla.

COMPONENTI

Materiali/strumenti:

- Braccio robot a 5 assi
- 4 servomotori
- Asse di legno (base d'appoggio)
- Viti
- Cavi
- Metro
- Cavo USB-A/micro USB-B
- Pezzi stampati in 3D
- Webcam

Software/programmi su PC:

- Fogli di Google
- Visual Studio Code
- Visual Studio

Librerie Python:

- Serial
- OpenCV
- Cv2
- Mediapipe
- Time

Schede elettroniche:

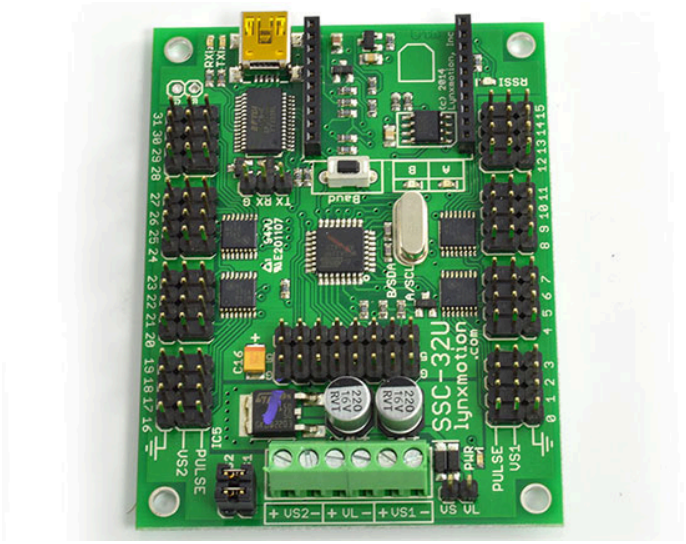
- SSC-32U

Alimentazione:

- Alimentatore da 45 Watt

SSC-32U

La scheda SSC-32U è un potente dispositivo progettato per il controllo di servomotori in una vasta gamma di applicazioni robotiche e di automazione. Capace di gestire fino a 32 servomotori. Dotata di connettori dedicati per ciascun servomotore, la SSC-32U semplifica il processo di collegamento e gestione dei dispositivi, consentendo una rapida integrazione in diverse configurazioni di sistema.
All'esterno della scheda si trova il connettore per la massa (GND), mentre al centro sono presenti i connettori per l'alimentazione (VS1 o VS2) e verso l'interno si trova lo slot per il segnale (PULSE).



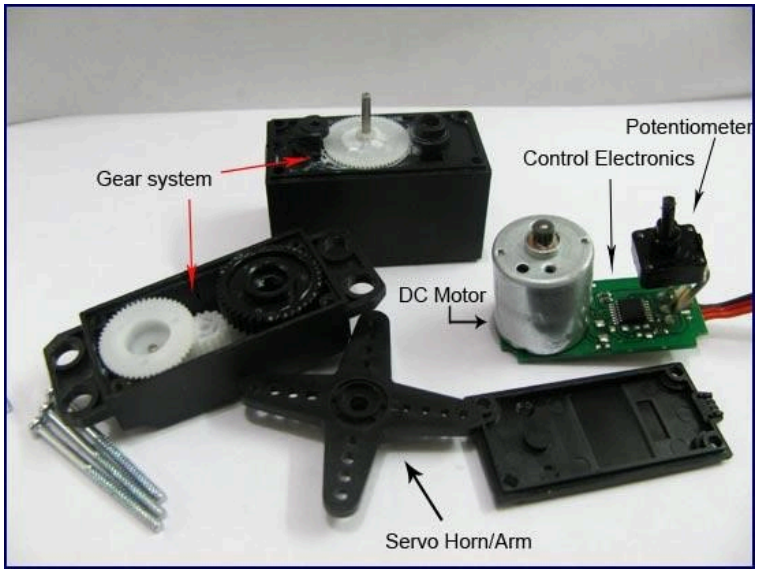
Questa disposizione logica facilita il cablaggio e l'organizzazione dei collegamenti. La scheda richiede due tipi di tensione per il funzionamento ottimale: una per la parte logica (VL) e una per alimentare i servomotori (VS1 per i servomotori da 0 a 15 e VS2 per quelli da 16 a 31). Questa suddivisione dell'alimentazione consente una gestione efficiente dell'energia e una maggiore flessibilità nell'implementazione dei progetti. Per la connessione e la programmazione, la SSC-32U è dotata di un connettore micro USB-B, che consente una facile comunicazione con un computer o altri dispositivi di controllo. Questo permette agli utenti di programmare la scheda e configurare i movimenti dei servomotori secondo questo formato di stringa:

#<ch>P<pw>T<time><cr>

· # rappresenta il carattere iniziale.
· <ch> è il numero del motore al quale si manda un impulso.
· P<pw> è la durata degli impulsi (che varia tra 500 µs e 2500 µs per portare un servomotore da 0° a 180°).
· T<time> il tempo necessario per compiere il movimento in µs.
· <cr> è il carattere ASCII 13, serve per segnalare la fine della stringa di comando.

SERVOMOTORI

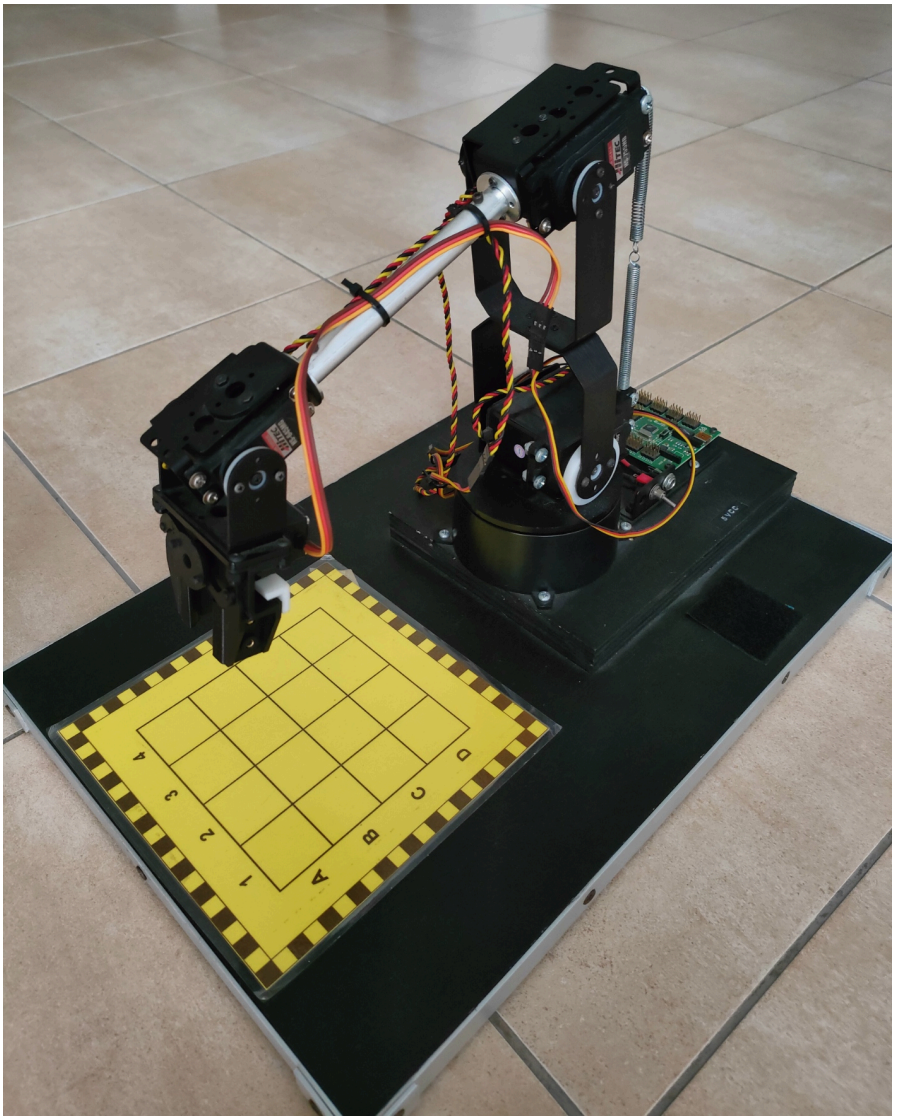
I servomotori sono una tipologia di motori in grado di ruotare un perno entro un angolo compreso tra 0° e 180°, mantenendo stabilmente la posizione raggiunta. Questa rotazione è controllata da un motore in corrente continua all'interno del dispositivo, supportato da un meccanismo di demoltiplica che incrementa la coppia di rotazione.
Il circuito di controllo interno del servomotore è fondamentale: utilizza un potenziometro resistivo per rilevare l'angolo di rotazione del perno e bloccare il motore nella posizione desiderata. Per comandare il motore, è necessario inviare un segnale digitale dal circuito di controllo. Questo segnale, di tipo impulsivo o PWM (Pulse Width Modulation), varia tra 500 µs e 2500 µs e rappresenta l'angolo di rotazione del perno, da 0° a 180°.
Sebbene i servomotori offrano una precisione limitata, sono ideali per applicazioni con robot di piccole dimensioni. Per progetti che richiedono maggiore precisione, come bracci robotici più complessi, si preferiscono solitamente motori passo-passo.
Nel mio progetto, i servomotori sono impiegati per controllare il manipolatore, rappresentando gli assi principali del movimento: la base, la spalla, il gomito e la pinza.



REALIZZAZIONE

Manipolatore

Il manipolatore robotico mostrato nella figura è un braccio articolato montato su una robusta base nera. La struttura del braccio comprende diversi giunti motorizzati, ciascuno azionato da servomotori, che consentono movimenti coordinati e articolati. I cavi elettrici collegati ai servomotori sono visibili lungo il braccio, indicando le connessioni necessarie per il controllo e l'alimentazione dei motori.
La base del manipolatore presenta una griglia numerata, utilizzata per facilitare il calcolo delle coordinate e il posizionamento preciso degli oggetti all'interno dell'area di lavoro. Questa griglia è essenziale per testare e calibrare il sistema, garantendo che i movimenti del braccio siano accurati e ripetibili.
Il manipolatore è progettato per essere controllato via software.



Taratura dei servomotori

La taratura dei servomotori è il processo mediante il quale si regola la posizione di riposo e l'intervallo di movimento del perno all'interno del servomotore. Questo è importante per garantire un funzionamento corretto e preciso del dispositivo.

Per eseguirla si utilizza il software Visual Studio e un foglio di Google. Su Visual Studio creo un programma che mi permetta di mandare un impulso, mediante la comunicazione seriale del PC, a ogni servomotore. L'interfaccia del programma è composta da trackbar che, variando il loro tra 500 e 2500, mi permettono di gestire gli impulsi da mandare a ogni singolo servomotore; nell'interfaccia troviamo anche delle caselle di testo dove viene riportato il valore della trackbar, in modo da poterlo annotare. Questo programma mi permetterà di capire a quali impulsi corrispondono i valori 0° e gli altri che mi permettono la regolazione del servo.

Qui è riportata l'interfaccia progettata da me per la taratura:

Dopodiché si passa sui fogli di Google per ricavare, mediante i dati ottenuti dal programma precedente, il grafico della retta del servomotore. Questo ci permetterà di ricavare l'equazione della retta, da cui otterremo il coefficiente angolare e l'intercetta con l'asse delle ordinate.

motore 0		
Posizione	Angolo	Impulsi
SX(P1)	-90	630
C	0	1530
DX(P2)	90	2450
q	1530	
m	10,11111111	

motore 2		
Posizione	Angolo	Impulsi
UP	90	1570
DOWN	180	770
q	2370	
m	-8,888888889	

motore 1		
Posizione	Angolo	Impulsi
C	0	600
UP	90	1430
q	600	
m	9,222222222	

motore 4	
OPEN	2000
CLOSE	1500

Equazione della retta:

Impulsi = angolo * m + q

- **m** è il coefficiente angolare della retta
- **q** è l'intercetta dell'ordinata.

Per ricavare m si utilizza il calcolo del rapporto tra la differenza degli estremi in ordinata e la differenza tra gli estremi in ascissa:

m = (P2y-P1y)/(P2x-P1x)

PROGRAMMA

```
import serial
import cv2
import mediapipe as mp
import time

# configurazione
debug = True  # Modalità debug attiva
flag = False  # Flag per la gestione del rilevamento delle mani

# Configurazione della porta seriale se debug è disabilitato
if not debug:
    ser = serial.Serial('COM3', 115200)
    ser.open()

tempo_iniziale_mani_multiple = None  # Tempo di rilevamento di più mani
tempo_flag = 0.5  # Soglia di tempo per il rilevamento delle mani
posizione_riposo_eseguita = False  # Flag per la posizione di riposo eseguita
tempo_senza_mani = None  # Tempo senza mani rilevate

# Parametri di configurazione per i servo
base_min = 20
base_mid = 90
base_max = 160
impbase = 0
mbase = 10.11
qbase = 630
cmdbase = '0'

spalla_min = 30
spalla_mid = 70
spalla_max = 90
impspalla = 0
mspalla = 9.22
qspalla = 850
cmdspalla = '0'

# usa il polso per controllare l'asse y
wrist_y_min = 0.3
wrist_y_max = 0.9

gomito_min = 20
gomito_mid = 90
gomito_max = 150
impgomito = 0
mgomito = -8.88
qgomito= 2400
cmdgomito = '0'

# usa la dimensione del palmo per controllare l'asse z
```

```
plam_size_min = 0.1
plam_size_max = 0.3

pinza_open_angle = 2000
pinza_close_angle = 1500
imppinza = 0
cmdpinza = '0'

cmdpolso = '0'

pugni = 0  # Contatore dei pugni rilevati

servo_angle = [base_mid, spalla_mid, gomito_mid, pinza_open_angle]  # Angoli iniziali dei servo
prev_servo_angle = servo_angle  # Angoli precedenti dei servo
fist_threshold = 7  # Soglia per il rilevamento del pugno

# Inizializzazione di MediaPipe
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands

# Inizializzazione della webcam
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

# Funzione per limitare un valore tra un minimo e un massimo
clamp = lambda n, minn, maxn: max(min(maxn, n), minn)

# Funzione per mappare un valore da un intervallo ad un altro
map_range = lambda x, in_min, in_max, out_min, out_max: abs((x - in_min) * (out_max - out_min) //
(in_max - in_min) + out_min)

# Controlla se la mano è chiusa
def is_fist(hand_landmarks, palm_size):
    # calcola la distanza tra il polso e la punta di ciascun dito
    distance_sum = 0
    WRIST = hand_landmarks.landmark[0]

    for i in [7, 8, 11, 12, 15, 16, 19, 20]:
        distance_sum += ((WRIST.x - hand_landmarks.landmark[i].x)**2 + \
                         (WRIST.y - hand_landmarks.landmark[i].y)**2 + \
                         (WRIST.z - hand_landmarks.landmark[i].z)**2)**0.5
    return distance_sum / palm_size < fist_threshold

# Converte le coordinate della mano in angoli per i servo
def landmark_to_servo_angle(hand_landmarks):
    servo_angle = [base_mid, spalla_mid, gomito_mid, pinza_open_angle]
    WRIST = hand_landmarks.landmark[0]
    INDEX_FINGER_MCP = hand_landmarks.landmark[5]

    # calcola la distanza tra il polso e l'indice
```

```
    palm_size = ((WRIST.x - INDEX_FINGER_MCP.x)**2 + (WRIST.y - INDEX_FINGER_MCP.y)**2 +
(WRIST.z - INDEX_FINGER_MCP.z)**2)**0.5

    if is_fist(hand_landmarks, palm_size):
        servo_angle[3] = pinza_close_angle
    else:
        servo_angle[3] = pinza_open_angle

    # calcola angolo base
    angle = WRIST.x
    servo_angle[0] = map_range(angle, 0, 1, base_max, base_min)

    # calcola angolo spalla
    wrist_y = clamp(WRIST.y, wrist_y_min, wrist_y_max)
    servo_angle[1] = map_range(wrist_y, wrist_y_min, wrist_y_max, spalla_max, spalla_min)

    # calcola angolo gomito
    palm_size = clamp(palm_size, plam_size_min, plam_size_max)
    servo_angle[2] = map_range(palm_size, plam_size_min, plam_size_max, gomito_max, gomito_min)

    servo_angle = [int(i) for i in servo_angle]

    return servo_angle

# Genera i comandi per portare il braccio robotico in posizione di riposo
def posizione_riposo():
    cmdbase = ('#' + '0' + 'P' + '1530' + 'T300' + '\r')
    cmdspalla = ('#' + '1' + 'P' + '1200' + 'T300' + '\r')
    cmdgomito = ('#' + '2' + 'P' + '1650' + 'T300' + '\r')
    cmdpinza = ('#' + '4' + 'P' + '2000' + 'T300' + '\r')
    cmdpolso = ('#' + '3' + 'P' + '1500' + 'T300' + '\r')
    return cmdspalla + cmdgomito + cmdbase + cmdpinza + cmdpolso

# Calcola la dimensione del palmo della mano
def calculate_palm_size(hand_landmarks):
    THUMB_MCP = hand_landmarks.landmark[2]
    PINKY_MCP = hand_landmarks.landmark[17]
    palm_width = ((THUMB_MCP.x - PINKY_MCP.x)**2 + (THUMB_MCP.y - PINKY_MCP.y)**2 +
(THUMB_MCP.z - PINKY_MCP.z)**2)**0.5
    return palm_width

# Funzione per inviare i comandi alla scheda SSC32U
def send_command(command):
    if not debug:
        ser.write(command.encode())
        time.sleep(0.4)
    else:
        print(command)

# Utilizza MediaPipe Hands per rilevare e tracciare le mani
with mp_hands.Hands(model_complexity=0, min_detection_confidence=0.5,
min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
```

```
        success, image = cap.read()

        if not success:
            print("Ignoring empty camera frame.")
            continue

        # Per migliorare le prestazioni della webcam
        image.flags.writeable = False
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
        results = hands.process(image)

        if results.multi_hand_landmarks:
            pugni = 0
            tempo_senza_mani = None

            for hand_landmarks in results.multi_hand_landmarks:
                palm_size = calculate_palm_size(hand_landmarks)
                if is_fist(hand_landmarks, palm_size):  # Controlla se la mano ha il pugno chiuso
                    pugni += 1

            if pugni > 1:  # Se ci sono due mani con i pugni chiusi
                flag = True

            if flag:  # Se sono state rilevate due mani con i pugni chiusi esegue le azioni per il braccio
robotico
                if len(results.multi_hand_landmarks) == 1:
                    hand_landmarks = results.multi_hand_landmarks[0]  # Utilizza solo le coordinate della
prima mano per il movimento del braccio
                    servo_angle = landmark_to_servo_angle(hand_landmarks)

                    if servo_angle != prev_servo_angle:
                        print("Servo angle: ", servo_angle)
                        prev_servo_angle = servo_angle

                        if not debug:
                            # calcoli degli impulsi da inviare alla scheda SSC32U
                            impbase = mbase * servo_angle[0] + qbase
                            impspalla = mspalla * servo_angle[1] + qspalla
                            impgomito = mgomito * servo_angle[2] + qgomito
                            imppinza = servo_angle[3]

                            cmdbase = ('#' + '0' + 'P' + str(round(impbase)) + 'T400' + '\r')
                            cmdspalla = ('#' + '1' + 'P' + str(round(impspalla)) + 'T400' + '\r')
                            cmdgomito = ('#' + '2' + 'P' + str(round(impgomito)) + 'T400' + '\r')
                            cmdpinza = ('#' + '4' + 'P' + str(round(imppinza)) + 'T400' + '\r')
                            cmdpolso = ('#' + '3' + 'P' + '1500' + 'T300' + '\r')

                            send_command(cmdspalla + cmdgomito + cmdbase + cmdpinza + cmdpolso)

                if len(results.multi_hand_landmarks) > 1:
                    if tempo_iniziale_mani_multiple is None:
                        tempo_iniziale_mani_multiple = time.time()
```

```
                tempo_trascorso = time.time() - tempo_iniziale_mani_multiple

                if tempo_trascorso > tempo_flag:
                    if not posizione_riposo_eseguita:
                        if not debug:
                            send_command(posizione_riposo())
                            print("Più di una mano rilevata")

                        posizione_riposo_eseguita = True

            if len(results.multi_hand_landmarks) < 2:
                tempo_iniziale_mani_multiple = None
                posizione_riposo_eseguita = False

            if len(results.multi_hand_landmarks) == 0:
                tempo_iniziale_mani_multiple = None

                if not debug:
                    send_command(posizione_riposo())
                    print("Nessuna mano rilevata")
        else:
            print("Nessuna o una sola mano con il pugno chiuso rilevata")

            if not debug:  # Se non sono state rilevate due mani con i pugni chiusi mantieni il braccio in
posizione di riposo
                send_command(posizione_riposo())
                print("Nessuna o una sola mano con il pugno chiuso rilevata")

    else:
        print("Nessuna mano rilevata")

        if tempo_senza_mani is None:
            # Imposta il tempo iniziale se è la prima volta che non ci sono mani rilevate
            tempo_senza_mani = time.time()
        else:
            # Calcola il tempo trascorso senza mani
            tempo_trascorso_senza_mani = time.time() - tempo_senza_mani

            # Se il tempo senza mani è maggiore di 0.5 secondi, esci dal ciclo della flag True
            if tempo_trascorso_senza_mani > tempo_flag and flag:
                flag = False

                if not debug:  # Se non sono state rilevate mani, mantieni il braccio in posizione di riposo
                    send_command(posizione_riposo())
                    print("Riattivazione necessaria")

    if results.multi_hand_landmarks is not None:  # Disegna sul frame le linee della mano
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
```

```
                mp_drawing_styles.get_default_hand_connections_style())

        image = cv2.flip(image, 1)

        # mostra l'angolo dei servo
        cv2.putText(image, str(servo_angle), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2,
cv2.LINE_AA)
        cv2.imshow('MediaPipe Hands', image)

        if cv2.waitKey(5) & 0xFF == 27:
            break

if not debug:
    ser.close()

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

Questo programma scritto in linguaggio Python è progettato per interagire con un braccio antropomorfo a 4 assi utilizzando le mani dell'utente rilevate tramite la webcam del PC. Sfrutta librerie specifiche come OpenCV per l'elaborazione delle immagini e MediaPipe per il rilevamento e il tracciamento delle mani.

Tuttavia, il programma non si limita semplicemente a controllare il braccio robotico. Integra una serie di controlli progettati per proteggere il manipolatore da comandi non voluti o da comportamenti imprevisti.

Innanzitutto, il braccio viene attivato solo quando vengono rilevate due mani con entrambi i pugni chiusi. Questa precauzione impedisce al braccio di muoversi accidentalmente durante il posizionamento iniziale delle mani davanti alla webcam.

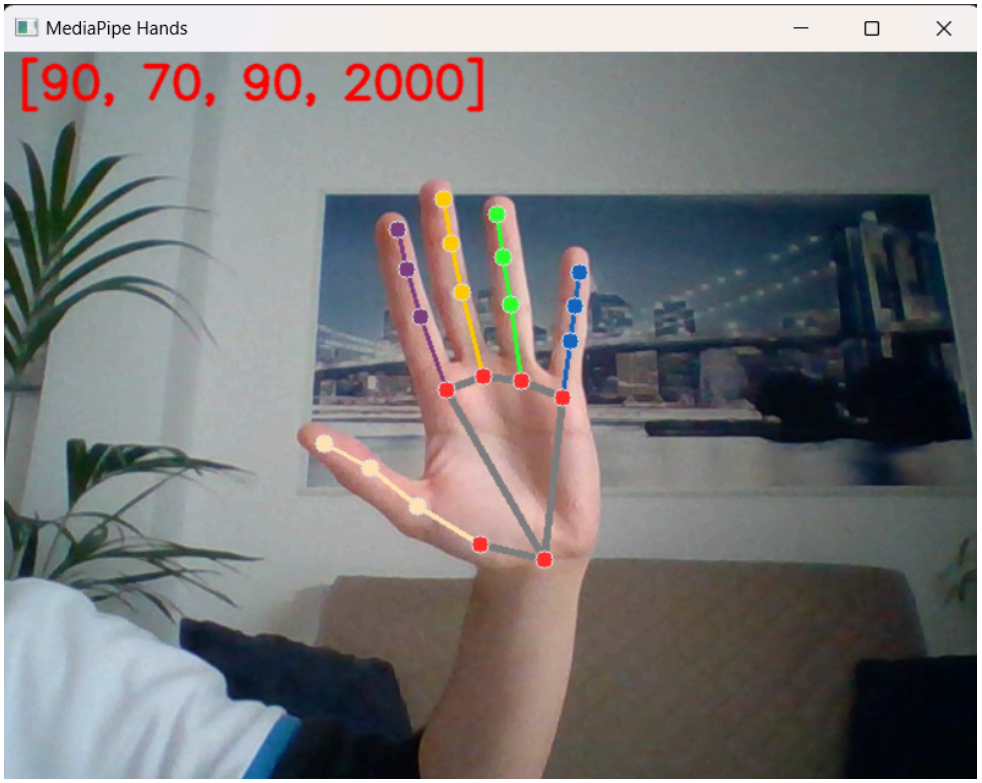
Una volta attivato, il braccio risponderà ai movimenti di una sola mano alla volta, seguendo gli assi di movimento descritti nell'introduzione. Questo evita conflitti tra i comandi delle mani e previene danni al braccio.

Per garantire ulteriormente la sicurezza, è stato implementato un secondo sistema: se più mani rimangono visibili per più di un secondo, il braccio entra in posizione di riposo. Questo tempo di attesa serve a evitare interferenze o conflitti di comando tra le mani, assicurando un funzionamento fluido e sicuro del manipolatore.

È importante notare che a causa delle limitazioni di potenza di calcolo del PC e delle caratteristiche non professionali dei servomotori utilizzati, il programma e il progetto potrebbero non essere sempre precisi al 100%. Tuttavia, questo serve principalmente a dimostrare le potenzialità della tecnologia e a fornire un punto di partenza per sviluppi futuri.

Il programma calcola gli impulsi necessari per il movimento del braccio e li invia tramite la porta seriale alla scheda di controllo SSC32-U. Inoltre, visualizza a schermo gli angoli previsti per ciascun servomotore, consentendo un monitoraggio visivo del funzionamento del braccio.
Per facilitare lo sviluppo e il debug, è stata inclusa una modalità di debug che consente di testare il codice senza dover necessariamente collegare il manipolatore, semplificando così il processo di sviluppo e debug del programma.

Ecco come si presenta la schermata dell'interfaccia dell'utente:



In alto a sinistra vengono mostrati gli angoli dei servomotori in tempo reale: base, spalla, gomito. Il quarto dato si riferisce allo stato della pinza, 2000 equivale ad aperta e 1500 a chiusa.
Le linee e i punti colorati che si possono vedere e che si sovrappongono alla mano sono dei marcatori che servono per svolgere i calcoli per il tracking.
Al termine dell'utilizzo il programma può essere arrestato tramite il tasto "Esc" della tastiera.

CONCLUSIONI

In conclusione, il progetto VisionHand Control rappresenta un'entusiasmante fusione tra tecnologia e creatività, portando l'interazione uomo-macchina a un livello completamente nuovo. Attraverso l'integrazione di servomotori controllati da un programma Python e il rilevamento dei movimenti delle mani tramite l'intelligenza artificiale, ho dimostrato come sia possibile controllare un manipolatore con gesti intuitivi e naturali. Ma questo è solo una piccolissima parte di quello che questa tecnologia può fare. L'obiettivo a cui ambisco è il controllo totale di un robot attraverso l'intero corpo, proprio come nel film "Real Steel", come menzionato nell'introduzione.

Per finire vorrei ringraziare i professori Arco Lorenzo e Porzio Giancarlo per avermi prestato il manipolatore.

BIBLIOGRAFIA E SITOGRAFIA

Bibliografia:

"L@borobotica volume B"

Sitografia:

https://www.laborobotica.com/le-tesine/

https://github.com/Crazycurly/gesture_MeArm

https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

https://github.com/google/mediapipe/blob/master/docs/getting_started/python.md

https://chat.openai.com/