



UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI INGEGNERIA ROBOTICA E DELL'AUTOMAZIONE

SISTEMI DI GUIDA E NAVIGAZIONE

RELAZIONE:
AHRS SENSOR-BASED
MOTION TRACKING GLOVE

Ilaria Martelli
Luca Ricci

26 maggio 2023

Indice

1 Contenuto dell'analisi e obiettivi	3
2 Descrizione del sistema	4
2.1 Materiale utilizzato	4
2.2 Caratterizzazione IMU	5
2.2.1 Dati accelerometro	5
2.2.2 Dati giroscopio	6
2.3 Caratterizzazione AHRS	8
2.3.1 Dati magnetometro	8
2.4 Posizionamento delle IMU	9
3 Implementazione hardware	10
3.1 Primo prototipo	10
3.2 Secondo prototipo	12
3.3 Versione finale	13
4 Software di basso livello per invio dati	14
4.1 Setup	14
4.2 Lettura dati	15
5 Il filtro	16
6 Rappresentazione virtuale	19
6.1 Pybullet/URDF	19
6.2 Unity	20
6.2.1 Elaborazione dei dati	20
7 Conclusioni	27

1 Contenuto dell'analisi e obiettivi

Lo scopo di questo progetto è la sintesi di un sistema di filtraggio dati di un complesso multi-IMU situato sul dito di un guanto indossabile per ricostruire la posa cinematica [2] e visualizzare una replica virtuale in tempo reale tramite Unity.

L'elaborato è diviso in 3 fasi principali:

1. lettura dei dati raccolti dalle IMU e spedizione tramite seriale (programma utilizzato: Arduino);
2. filtro che elabora i dati ricevuti da IMU per restituirne l'assetto (linguaggio utilizzato: Python);
3. rappresentazione tridimensionale di una mano, che utilizza le misure di assetto trovate per posizionarsi in modo da seguire i movimenti del guanto (programma utilizzato: Unity).

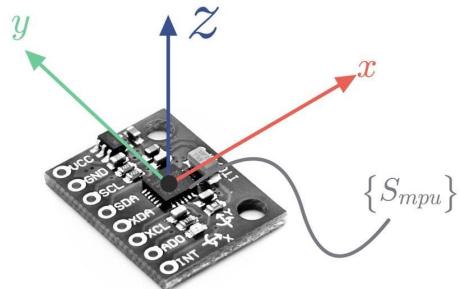


Figura 1: Sistemi di riferimento *body* su una IMU.

2 Descrizione del sistema

2.1 Materiale utilizzato

Per il progetto sono state utilizzate:

1. una MPU-6050 a 6 gradi di libertà (*3dof accelerometer + 3dof gyroscope*);
2. due MPU-9250 a 9 gradi di libertà (*3dof accelerometer + 3dof gyroscope + 3dof magnetometer*);
3. un multiplexer TCA9548A (per permettere l'utilizzo di più IMU sullo stesso canale I2C);
4. una scheda ESP32 programmabile con Arduino.

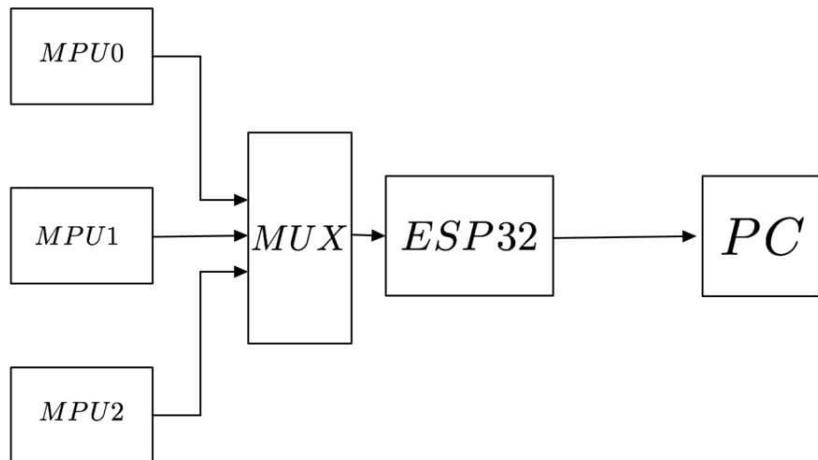


Figura 2: Schema a blocchi complessivo.

2.2 Caratterizzazione IMU

Una IMU è un'unità di misura inerziale che comprende tre accelerometri e tre giroscopi posti sui tre assi di interesse dell'IMU (assi *body*).

2.2.1 Dati accelerometro

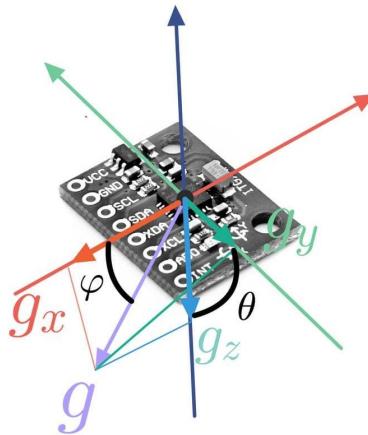


Figura 3: Accelerometro.

Attraverso i dati dell'accelerometro, si può inferire l'inclinazione di roll (rotazione attorno all'asse x) e pitch (rotazione attorno all'asse y) dell'IMU, considerando i valori trovati sui tre assi locali (*body*) come le componenti parziali del vettore gravità, che sappiamo puntare verso il basso in assi fissi (*navigation*).

$$\phi_a = \tan^{-1} \left(\frac{a_y}{a_z} \right) \quad (1)$$

$$\theta_a = \tan^{-1} \left(\frac{-a_x}{a_y \sin \phi_a + a_z \cos \phi_a} \right) \quad (2)$$

D'altra parte, usare solo le misure date dagli accelerometri genera errori nel caso in cui l'IMU, invece che venire ruotata, sia sottoposta a un'accelerazione data da un moto lineare: in tal caso, questo metodo riporterebbe rotazioni che non esistono. Per questo è necessario filtrare le misure con altri sensori, in modo che gli errori di uno vengano compensati dagli altri (la cosiddetta *sensor fusion*).

2.2.2 Dati giroscopio

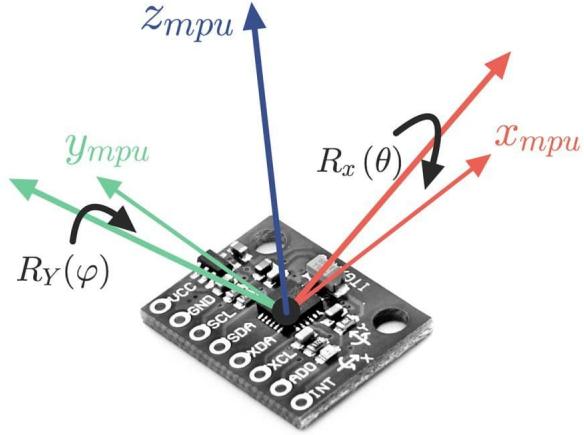


Figura 4: Giroscopio.

Allo stesso modo dell'accelerometro, il giroscopio può trovare la velocità angolare sui tre assi di riferimento. In questo caso, una semplice operazione di integrazione ci consente di ricavare gli angoli relativi nel tempo, che poi vengono sommati alla loro misura precedente. Questo metodo necessita la conoscenza dell'orientazione iniziale dell'IMU, ma tale richiesta viene soddisfatta ponendo gli angoli iniziali a 0 e facendo partire il programma che invia i dati quando il guanto è parallelo al suolo (roll e pitch nulli).

$$\dot{q}_{gyro} = \frac{1}{2} \Omega_t q$$

$$q_{gyro} = q + \dot{q}_{gyro} dt$$

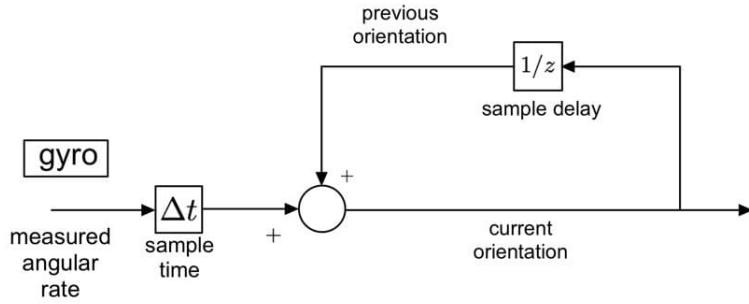


Figura 5: Elaborazione dati giroscopio.

Per l'integrazione delle misure delle velocità angolari, sono state usate le formule (3), (4) e (5), che combinano sia la semplice operazione di integrazione con una rotazione rispetto agli angoli dello step precedente, in modo che i sistemi di riferimento tra angoli trovati con accelerometro-magnetometro e giroscopio siano allineati.

$$\dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta \quad (3)$$

$$\dot{\theta} = q \cos \phi - r \sin \phi \quad (4)$$

$$\dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta \quad (5)$$

Un problema più grave invece è la considerazione dell'errore del giroscopio nel tempo: poiché la misura della velocità angolare è sottoposta a rumore, integrare tale misura contribuisce al *drifting*, ovvero una forma di alterazione della misura in cui l'angolo ricavato aumenta nel tempo anche se l'IMU rimane ferma.

Per risolvere il *drifting* si usa la misura ricavata dall'accelerometro come correzione.

2.3 Caratterizzazione AHRS

Un AHRS (talvolta chiamato MARG per *Magnetic, Angular Rate and Gravity*) è un sistema di riferimento di orientazione e direzione che include una IMU e un magnetometro.

In particolare, le MPU-9250 usate hanno come base un'IMU uguale alle MPU-6050 a cui è stato aggiunto un magnetometro AK8963 a 3 gradi di libertà.

2.3.1 Dati magnetometro

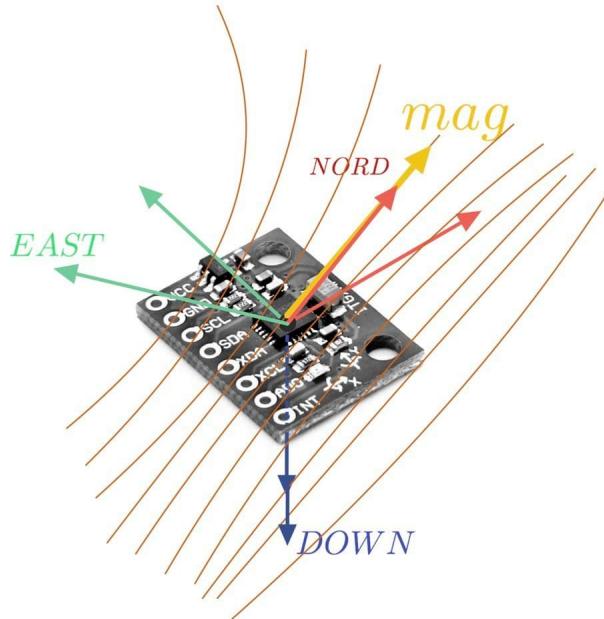


Figura 6: Magnetometro.

Nelle AHRS ad accelerometro e giroscopio delle IMU si aggiunge la misura di *heading* data dal magnetometro, che permette di ricavare anche lo yaw (rotazione attorno all'asse z) del dispositivo.

I passaggi sono i seguenti: si ruotano i dati ricevuti dal magnetometro (m) utilizzando i valori di roll e pitch già ricavati dall'accelerometro, e da essi si ricava lo yaw.

$$\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = R_y(\theta_a)R_x(\phi_a) \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix}$$

$$\psi_m = \tan^{-1} \left(\frac{-b_y}{b_x} \right)$$

Da cui risulta

$$\psi_m = \tan^{-1} \left(\frac{m_x \sin \phi_a - m_y \cos \phi_a}{m_x \cos \theta_a + m_y \sin \theta_a \sin \phi_a + m_z \sin \theta_a \cos \phi_a} \right) \quad (6)$$

2.4 Posizionamento delle IMU

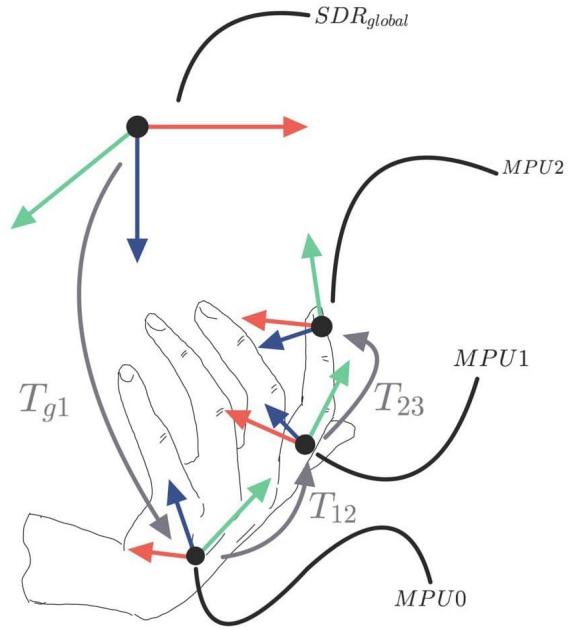


Figura 7: Sistemi di riferimento del dito.

Le MPU-9250, essendo provviste di magnetometro per misurare l'*heading*, sono state poste una sul dorso della mano (MPU0), per poterne visualizzare lo yaw, e una sulla falange prossimale, ovvero quella più vicina al palmo (MPU1), in modo da poter visualizzare rotazioni relative tra il dito e il palmo della mano. La MPU-6050 è stata invece posizionata sulla falange distale, ovvero quella più vicina alla punta del dito (MPU2).

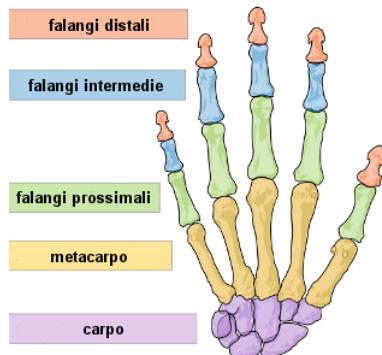


Figura 8: Falangi di una mano.

3 Implementazione hardware

La fase di sviluppo di questo progetto ha subito molti problemi nella parte hardware, portando a vari prototipi e infine alla riduzione della complessità del progetto stesso.

Infatti si è passati da un sistema di sensorizzazione completa di una mano, con 2 IMU per dito più 3 IMU per modellare il movimento del palmo (in particolare il piegamento sul metacarpo dato dal pollice e quello dato dal mignolo), per un totale di 13 IMU e 2 multiplexer, a un progetto semplificato su un singolo dito, per un totale di 3 IMU e 1 multiplexer.

3.1 Primo prototipo



Figura 9: V1

Inizialmente si era pensato di utilizzare un filo conduttivo per collegare insieme i dispositivi posti sul guanto: questo avrebbe consentito di cucire più saldamente i vari componenti, utilizzando sia cuciture normali che il filo conduttivo stesso, e allo stesso tempo avrebbe dato al guanto un aspetto più *tight fit*, non avendo cavi che si estendono oltre il volume del guanto.

Purtroppo questa strada ha portato a molti problemi di interferenze tra circuiti troppo vicini, e ancor più grave la presenza di cortocircuiti, tra cui almeno uno sui molteplici fili che collegavano i pin VCC e GND per l'alimentazione delle IMU, causando drastici cali di tensione che permettevano l'accensione di appena 2/3 IMU su 12 (sparse, non sullo stesso dito).

Per cercare di risolvere il problema si è ricorsi all'uso di un multmetro per l'individuazione dei cortocircuiti, e una melassa isolante da spalmare sui fili. Siccome anche questo non ha portato ai risultati sperati, si è deciso di ritentare modificando il progetto.

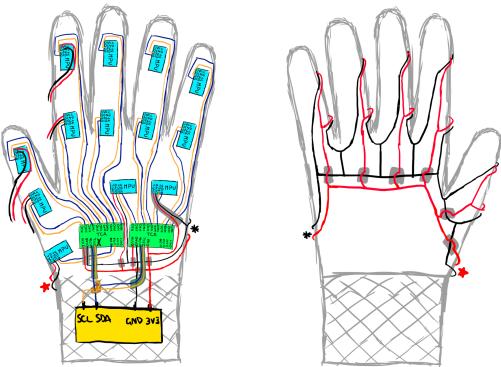


Figura 10: V1 - schema dei collegamenti

3.2 Secondo prototipo

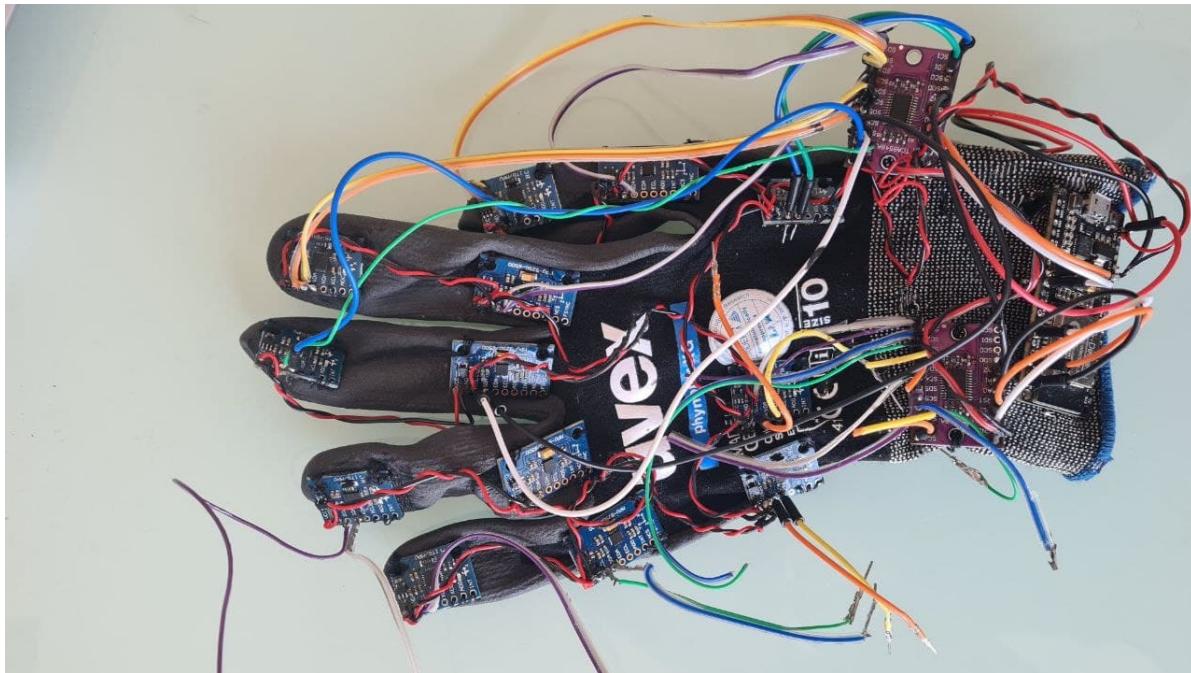


Figura 11: V2

Per il secondo prototipo del guanto si è abbandonata l’idea del filo per cucire conduttivo, sostituendolo con dei normali cavetti isolati, e in tal modo si è riusciti ad arrivare in una configurazione tale per cui tutte le IMU si accendessero senza problemi.

Ci sono stati tuttavia altri problemi imprevisti: alcuni collegamenti I2C perdevano sporadicamente segnale anche per minimi movimenti del guanto, e il programma di recezione dati riceveva zeri su tutte le misure dell’IMU; i cavi inoltre sono stati saldati e crimpati per avere collegamenti stabili ed evitare ulteriori errori, ma per effettuare operazioni di debugging è stato necessario tagliarli e ricollegarli più volte; infine, molte IMU usate avevano problemi di inizializzazione e su alcuni sensori restituivano il valore di fondoscala invece che la misura effettiva, facendo pensare dopo vari tentativi di debugging falliti che fosse l’IMU stessa ad essere non funzionante.

Per questi motivi si è passati a una terza versione del guanto, e poiché la probabilità di incorrere in problemi hardware aumenta proporzionalmente rispetto al numero di dispositivi utilizzati, si è quindi ritenuto necessario diminuire la complessità del progetto.

3.3 Versione finale

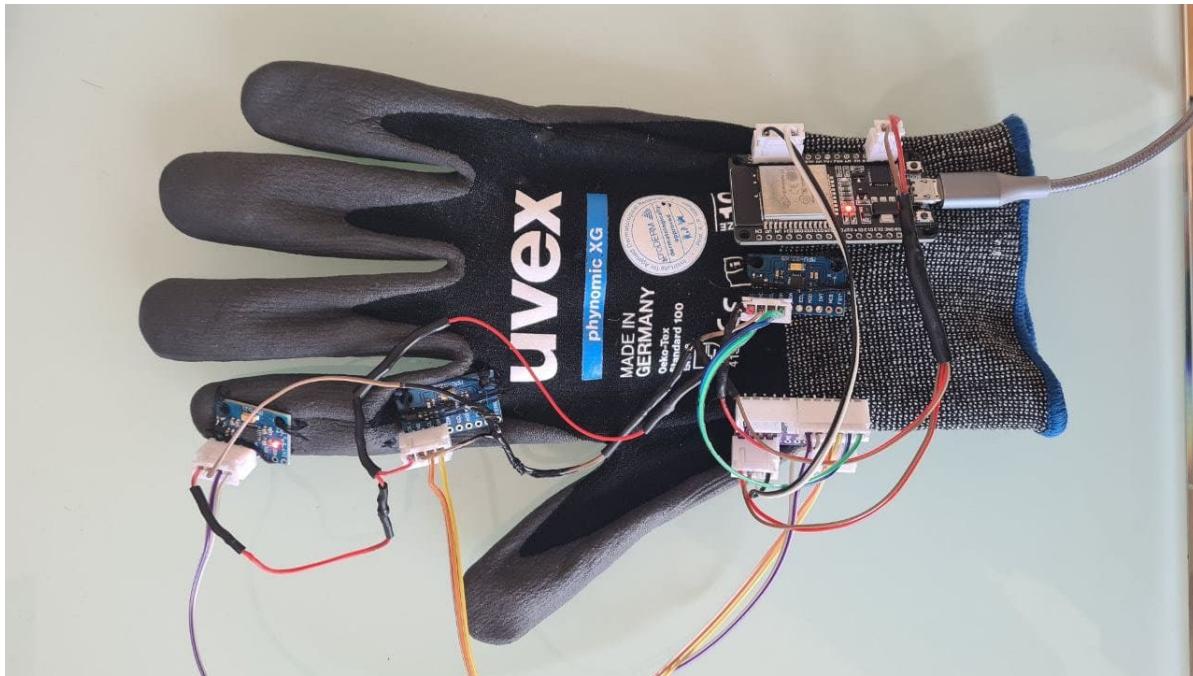


Figura 12: V3 - semplificato

La versione finale è quindi focalizzata su un solo dito, l'indice, e di conseguenza impiega molti meno dispositivi.

Tutti i dispositivi utilizzati sono stati testati in modo approfondito prima ancora di essere installati sul guanto, per assicurarsi che funzionasse correttamente.

Con questa versione non ci sono stati problemi, e ogni nuova IMU collegata è stata testata prima di connettere la successiva. I cavi invece di venire saldati direttamente sono stati collegati con degli header (JTS). L'inserzione degli header JTS garantisce un collegamento solido, evitando problemi di connessioni instabili (molto presenti nel caso di I2C), ma allo stesso tempo consente di poterli scollegare e ricollegare in qualsiasi momento.

4 Software di basso livello per invio dati

La scheda ESP32 è stata programmata da Arduino: non sono state utilizzate librerie (a parte ovviamente `Wire.h`), ma ci si è rifatti a [7] e [1] come riferimento per la costruzione del programma.

Di seguito una breve panoramica del funzionamento.

4.1 Setup

Il programma inizia facendo un check di tutti i canali del multiplexer e inizializzando le IMU presenti.

Poiché i giroscopi potrebbero misurare angoli diversi da 0 all'avvio, è stata implementata la possibilità di calibrarli tramite la funzione `calc_gyro_offset()`, che richiede di lasciare l'IMU immobile per raccogliere misure di velocità angolari di cui poi verrà fatta la media, che sarà il bias che viene poi sottratto a ogni successiva lettura; questa funzione è attivabile e disattivabile tramite la variabile booleana `calibrate_gyro`.

Durante il setup viene mandato un segnale al magnetometro con la funzione `initAK8963()`, e se l'IMU in questione è una 9250 ad esso segue l'inizializzazione del magnetometro, mentre se è una 6050 si interrompe senza fare nulla.

Per le MPU-9250 è stata anche implementata la possibilità di calibrare i magnetometri tramite la funzione `calc_magn_hard_iron_soft_iron_offset()` per eliminare i bias soft-iron, dati da interferenze di materiali magnetici, e hard-iron, date da parti che generano un proprio campo magnetico presenti sulla scheda stessa (basti pensare al campo magnetico prodotto dalla corrente che scorre in un circuito); questa funzione è attivabile e disattivabile tramite la variabile booleana `calibrate_magn`.

Riferimenti sulla calibrazione del magnetometro in [1], [5] e [6].

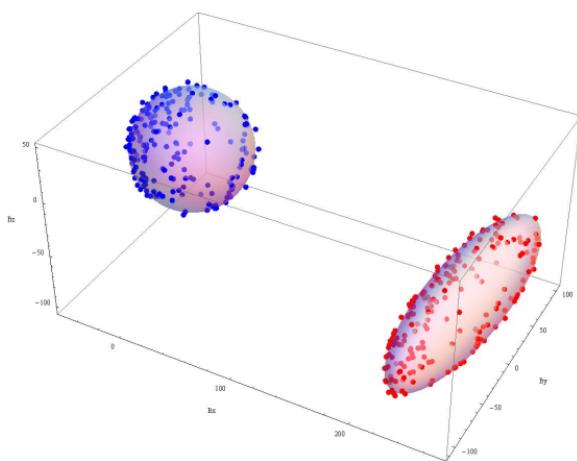


Figura 13: Campo magnetico corretto (sférico e centrato in $\bar{0}$) a sinistra, a confronto con campo magnetico distorto (bias hard-iron lo decentrano e bias soft-iron lo deformano) a destra.

Un'ultima funzione, normalmente disattivata, `I2C_scan()`, permette di effettuare uno scan di tutti i possibili address sul canale I2C per controllare quali dispositivi rispondono per ogni canale del multiplexer: tale funzione è stata particolarmente utile nelle fasi di sviluppo e debugging del progetto.

4.2 Lettura dati

Nel loop principale i dati vengono letti e stampati su seriale a ogni iterazione.

La funzione `read_data()` (che richiama la sotto-funzione `read_raw_data()`) consente di accedere ai registri di memoria delle IMU e ricostruire la giusta lettura, prima unendo i byte raw (high e low), e poi correggendo con apposite variabili date su datasheet o con le misure di bias trovate se sono state effettuate calibrazioni nella fase di setup.

La lettura dei dati delle IMU avviene a una frequenza di 400 kHz complessivi e quindi ogni IMU viene letta a circa 130 kHz. Ogni dato da leggere è composto da 8 bit e sono presenti 9 misure da leggere, conseguentemente 72 bit per ogni MPU.

La velocità del canale I2C risulta sufficiente per avere un buon aggiornamento della stima e un delay ridotto.

La funzione `print_data()` invece invia al seriale i dati letti, in modo che possano essere recepiti da PC ed elaborati nel filtro.

5 Il filtro

Per l'elaborazione dei dati ricevuti dalle IMU è stato utilizzato il filtro complementare, che unisce un'implementazione relativamente semplice a dei risultati di performance comparabili a filtri più complessi, come il filtro di Kalman.

Esso si basa su un processo di *sensor fusion* con il raggiungimento di una stima degli angoli di roll, pitch e yaw tramite due diversi approcci, per poi unire i due contributi a meno di una costante complementare α che esprime il grado di fiducia nel corrispettivo sensore.

In particolare, nel progetto in questione si è fatto anche riferimento all'articolo [3], che applica un filtro complementare leggermente diverso da quello canonico.

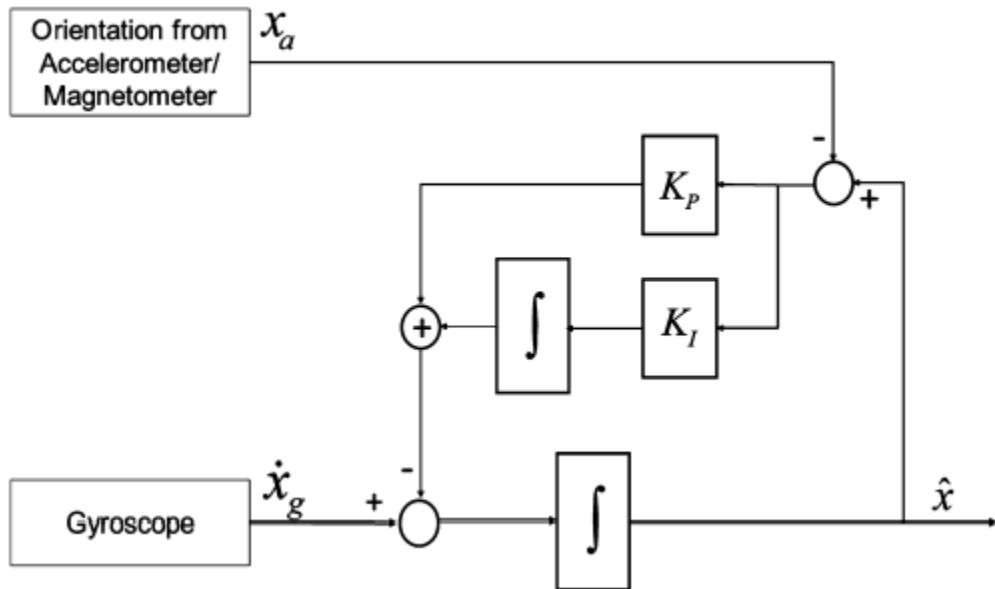


Figura 14: Schema filtro complementare proposto

Nella prima parte, gli angoli vengono ottenuti dall'integrazione delle misure del giroscopio, paragrafo (2.2.2) equazioni (3) (4) e (5):

$$\begin{cases} \dot{\phi} = p + q \sin \phi \tan \theta + r \cos \phi \tan \theta \\ \dot{\theta} = q \cos \phi - r \sin \phi \\ \dot{\psi} = q \sin \phi \sec \theta + r \cos \phi \sec \theta \end{cases}$$

nella seconda parte, roll e pitch derivano da formule basate su considerazioni sulla gravità già discusse nel paragrafo (2.2.1) equazioni (1) e (2); lo yaw, come già discusso nel paragrafo (2.3.1), necessita di un riferimento di *heading* e quindi del magnetometro, paragrafo (2.3.1)

equazione (6):

$$\begin{cases} \phi_a = \tan^{-1} \left(\frac{a_y}{a_z} \right) \\ \theta_a = \tan^{-1} \left(\frac{-a_x}{a_y \sin \phi_a + a_z \cos \phi_a} \right) \\ \psi_m = \tan^{-1} \left(\frac{m_x \sin \phi_a - m_y \cos \phi_a}{m_x \cos \theta_a + m_y \sin \theta_a \sin \phi_a + m_z \sin \theta_a \cos \phi_a} \right) \end{cases}$$

Infine quindi tali stime basate sui singoli sensori vengono unite: la formula classica ([4], quaternioni) è

$$\hat{q} = \alpha q_g + (1 - \alpha) q_{am} \quad (7)$$

che in seguito è stata ripresa ed espansa ([3]) come

$$\hat{x}_{new} = \frac{1}{s} \left[\dot{x}_g + (K_p + \frac{K_I}{s})(x_{am} - \hat{x}) \right] \quad (8)$$

e che rivista e semplificata ($K_I = 0$) diventa

$$\hat{x}_{new} = \frac{1}{s} \left[\alpha \dot{x}_g + (1 - \alpha)(x_{am} - \hat{x}) \right] \quad (9)$$

dove

$$\dot{x}_g = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}, \quad x_{am} = \begin{bmatrix} \phi_a \\ \theta_a \\ \psi_m \end{bmatrix} \quad (10)$$

\hat{x}_{new} è la stima finale degli angoli di roll pitch e yaw, \hat{x} è la stima allo step precedente, \dot{x}_g sono le velocità angolari ricavate dai giroscopi dopo le correzioni di rotazione, e x_{am} sono le misure ricavate da accelerometro e magnetometro.

La formula finale quindi risente di entrambi i metodi, dove la variabile proporzionale K_p è stata riconvertita in α .

È stato poi notato sperimentalmente che il valore migliore per α fosse più vicino a 1, e si assestasse su 0.9.

Le funzioni di filtraggio sono presenti nel codice python, in particolare il lavoro viene svolto dalla funzione `Complementary2eul()`.

```

def Complementary2Eul(asset,acc,gyro,mag,dt):
    '''complementary filter based on accelerometer,gyro,mag mesures using euler angles XYZ'''

    alpha = 0.9
    phi = asset[0]
    theta = asset[1]
    psi = asset[2]
    a_x = acc[0]
    a_y = acc[1]
    a_z = acc[2]
    g_x = gyro[0]
    g_y = gyro[1]
    g_z = gyro[2]

    p = g_x
    q = g_y
    r = g_z

    phi_dot_gyro = p + q * np.sin(phi) * np.tan(theta) + r * np.cos(phi) * np.tan(theta)
    theta_dot_gyro = q * np.cos(phi) - r * np.sin(phi)
    psi_dot_gyro = q * np.sin(phi) * np.cos(theta)**-1 + r * np.cos(phi) * np.cos(theta)**-1

    phi_acc = np.arctan2( a_y , a_z )
    theta_acc = np.arctan2(-a_x,a_y*np.sin(phi_acc)+a_z*np.cos(phi_acc))
    psi_acc = np.arctan2(np.sqrt(a_y**2+a_x**2),a_z)

    if all(mag) != 0:
        m_x = mag[0]
        m_y = mag[1]
        m_z = mag[2]
        m_vect = np.array([[m_x],[m_y],[m_z]])
        #
        R = np.dot(Ry(theta_acc), Rx(phi_acc))
        b = np.dot(R, m_vect)
        b_x, b_y = b[0,0], b[1,0]
        psi_mag = np.arctan2(-b_y, b_x)
    else:
        psi_mag= 0

    gyro = np.array( [ phi_dot_gyro * dt ,
                      theta_dot_gyro * dt ,
                      psi_dot_gyro * dt ] )

    accmag = np.array( [phi_acc,theta_acc,psi_mag] )
    asset += alpha * gyro + (1-alpha) * (accmag - asset)
    return asset

```

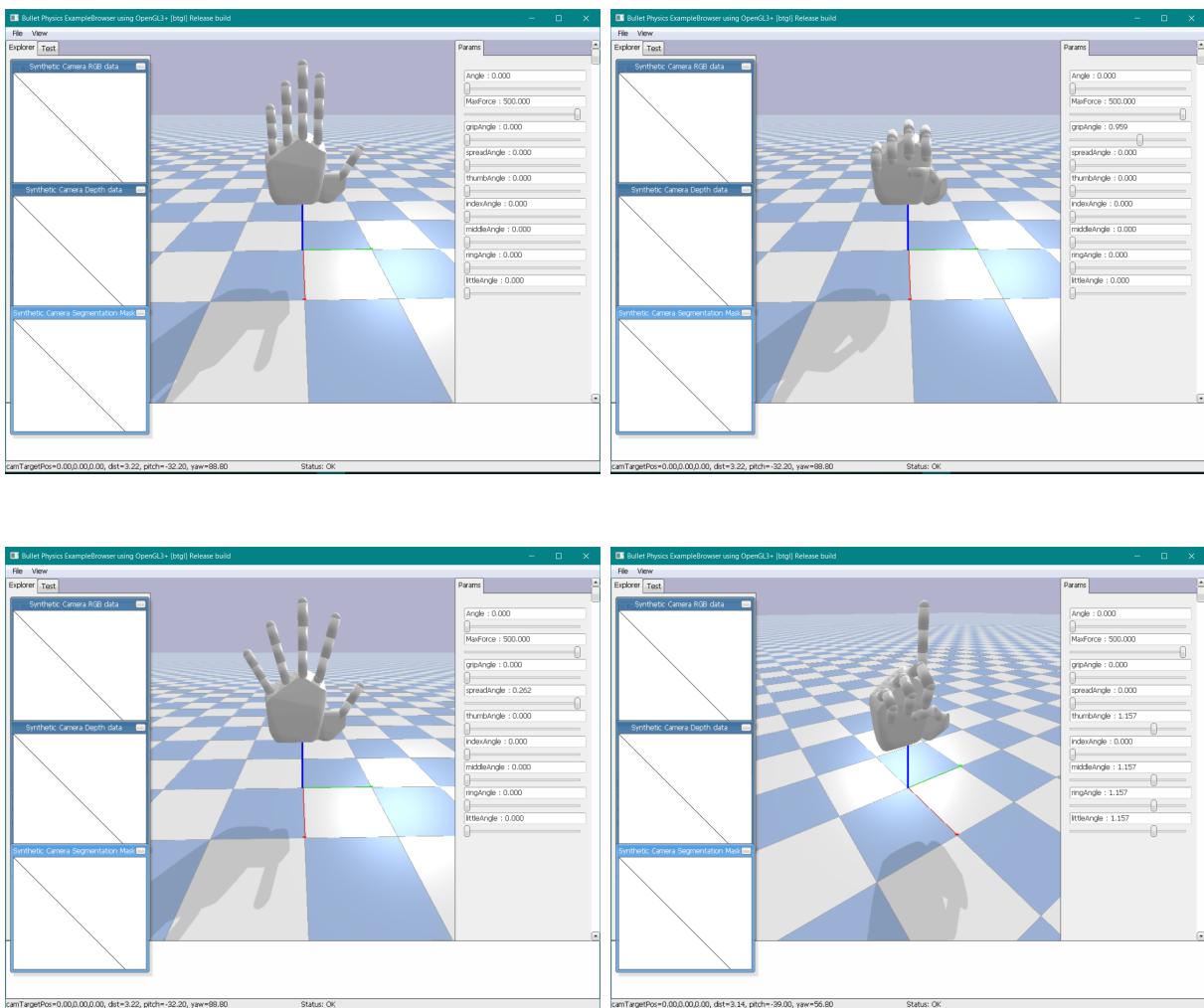
Figura 15: Codice del filtro complementare.

6 Rappresentazione virtuale

6.1 Pybullet/URDF

Inizialmente si era pensato di utilizzare la libreria Python Pybullet per creare l'ambiente tridimensionale, e un file .urdf.xacro per generare la mano.

Qui vengono riportate alcune immagini di tale prototipo virtuale.



6.2 Unity

In seguito si è passati all'utilizzo di Unity e alla mano virtuale dell'Oculus Rift.

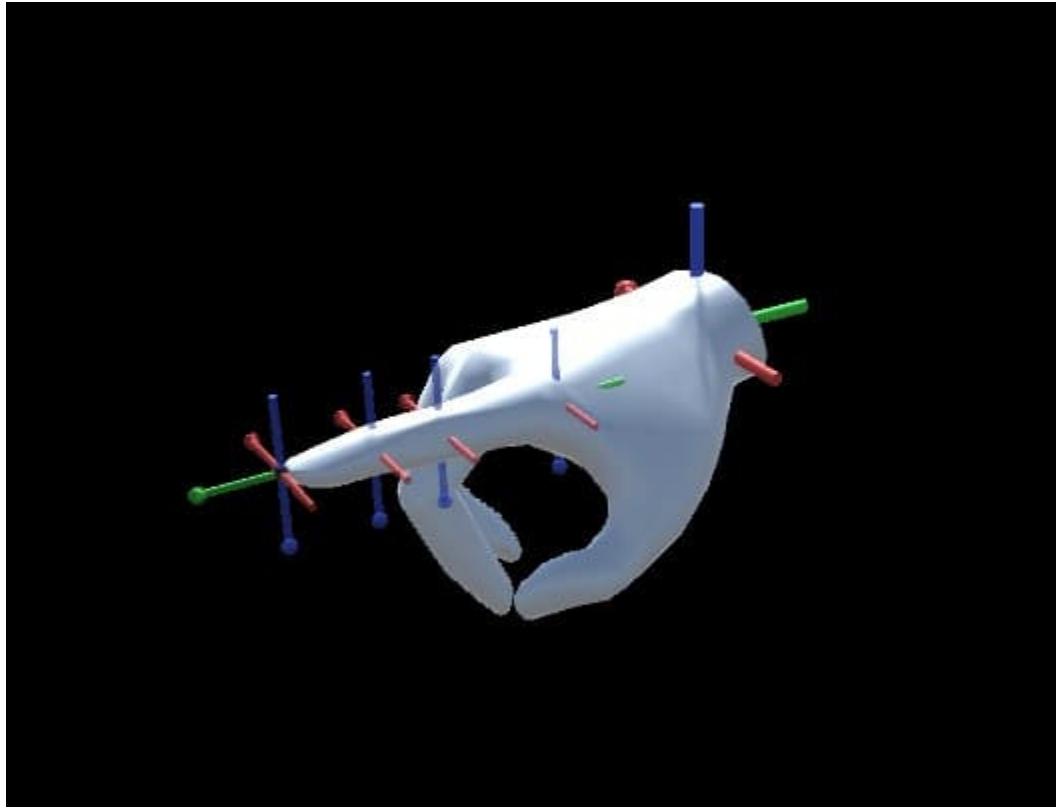


Figura 18: Visualizzazione della mano su Unity

6.2.1 Elaborazione dei dati

Una volta filtrati i dati tramite la batteria di stimatori d'assetto, vengono inviate le stime a una batteria di stimatori di orientazione relativa. Essi hanno lo scopo di calcolare l'angolo di giunto delle falangi della mano prendendo in ingresso l'assetto di 2 link successivi e calcolandone la rotazione relativa.

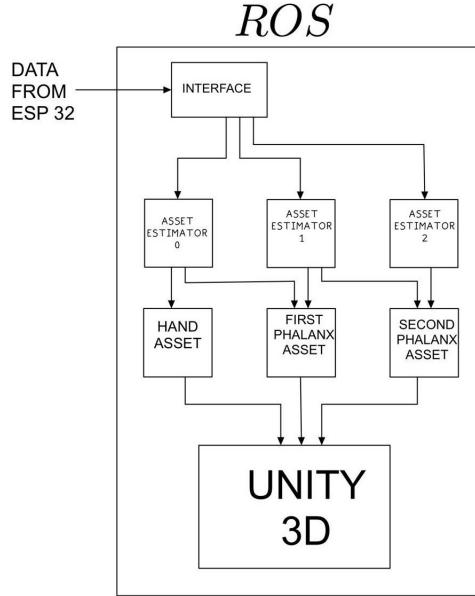


Figura 19: Schema di stima e di calcolo di rotazioni relative

$$T_{01_{rel}} = T_0(\theta_0, \phi_0, \psi_0)^{-1} \cdot T_1(\theta_1, \phi_1, \psi_1) \quad (11)$$

$$T_{12_{rel}} = T_1(\theta_1, \phi_1, \psi_1)^{-1} \cdot T_2(\theta_2, \phi_2, \psi_2) \quad (12)$$

Dalla matrice di rotazione relativa possiamo estrapolare gli angoli di Eulero:

$$R = T_{rel} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \quad (13)$$

$$\text{phi} = \text{atan2}(r_{23}, r_{33}) \quad (14)$$

$$\text{theta} = -\arcsin(r_{13}) \quad (15)$$

$$\text{psi} = \text{atan2}(r_{12}, r_{11}) \quad (16)$$

Una volta calcolati gli angoli di giunto vengono inviati tramite ROS2 i messaggi relativi ai singoli angoli di giunto e all'assetto della mano completa a un software di visualizzazione grafica.

Per validare i risultati è stata impostata una simulazione su Unity3D che ci permette di visualizzare in real time i risultati della stima d'assetto tramite codice C# integrato in Unity.

Una volta che le stime degli angoli di giunto erano a disposizione su unity, i valori sono stati trasformati in quaternioni (i software di grafica 3D ne fanno uso estensivo visto il minor costo computazionale) e tra un campione e il successivo viene eseguita una interpolazione SLERP (spherical linear interpolation) di velocità programabile.

Essa è particolarmente indicata per interpolare grandezze di tipo angolare e migliora il risultato rendendo l'animazione più fluida e meno rumorosa.

$$q = \begin{bmatrix} \cos(\frac{\phi}{2}) \cos(\frac{\theta}{2}) \cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2}) \sin(\frac{\theta}{2}) \sin(\frac{\psi}{2}) \\ \sin(\frac{\phi}{2}) \cos(\frac{\theta}{2}) \cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2}) \sin(\frac{\theta}{2}) \sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2}) \sin(\frac{\theta}{2}) \cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2}) \cos(\frac{\theta}{2}) \sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2}) \cos(\frac{\theta}{2}) \sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2}) \sin(\frac{\theta}{2}) \cos(\frac{\psi}{2}) \end{bmatrix} \quad (17)$$

$$Slerp(q_1, q_2, u) = q_1 \cdot (q_1^{-1} \cdot q_2)^u \quad (18)$$

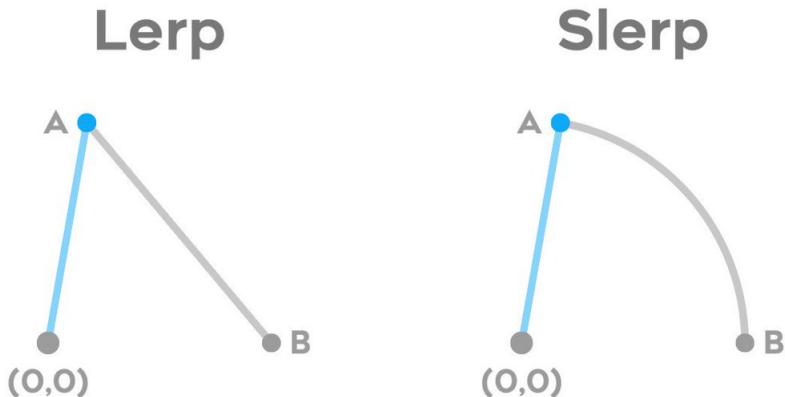


Figura 20: lerp - slerp piano

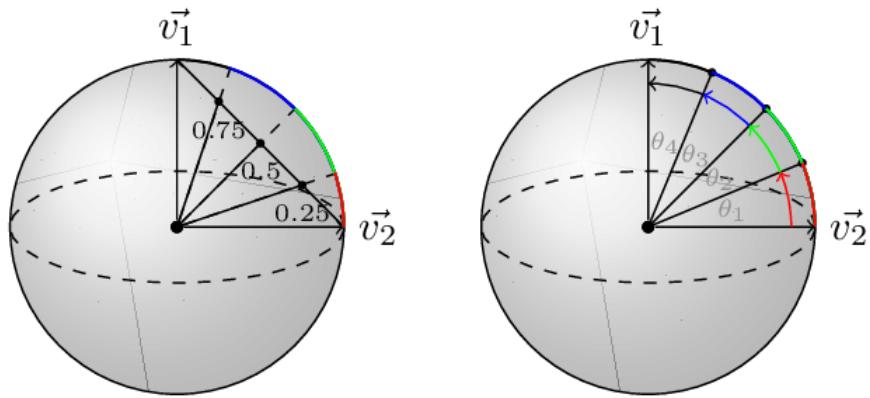


Figura 21: lerp - slerp differenza sulla sfera del quaternione unitario

È stata importata una mesh 3D di una mano da Oculus Rift, e ad essa sono stati agganciati sistemi di riferimento coerenti con quelli delle IMU. I sistemi di riferimento vengono visualizzati secondo lo schema RGB - XYZ e sono solidali con i giunti presenti nella mano virtuale.

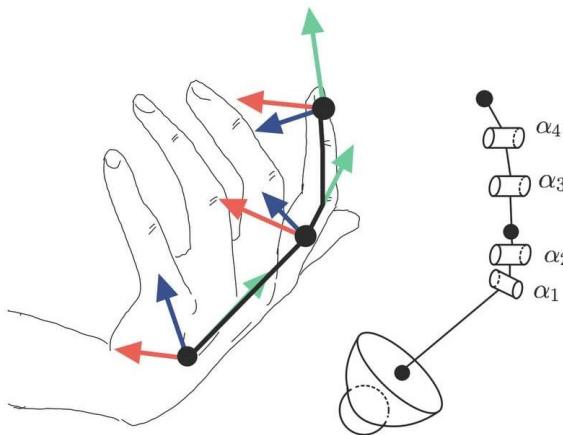
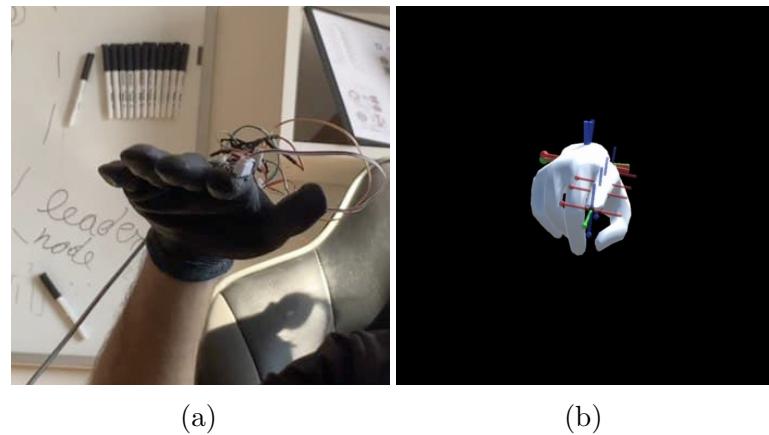


Figura 22: Catena cinematica.

Per la validazione dei risultati di stima abbiamo fatto uso esclusivamente di considerazioni di *motion tracking*: muovendo il guanto, abbiamo considerato buona la stima quando la mano virtuale ne seguiva il movimento.

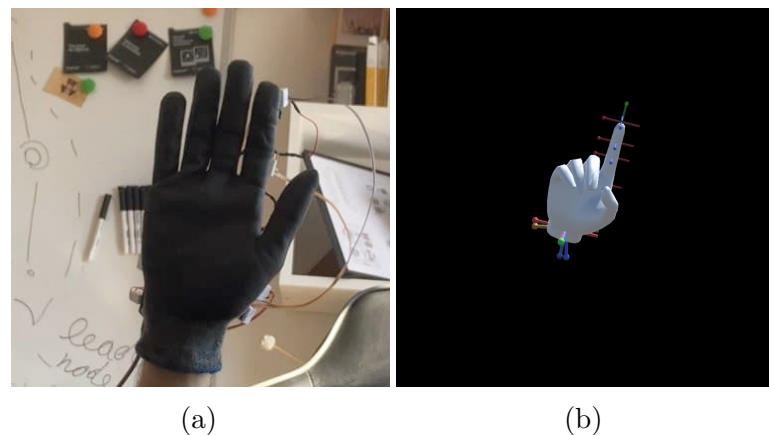
Per via delle convenzioni utilizzate al filtro e quelle usate da Unity la validità della simulazione risiede tra angoli compresi tra 0 e 180 gradi. Sforando questi angoli si hanno dei cambi di orientazione della mano. Data la stima eccessivamente imprecisa dell'angolo α_1 esso è stato mantenuto a 0.



(a)

(b)

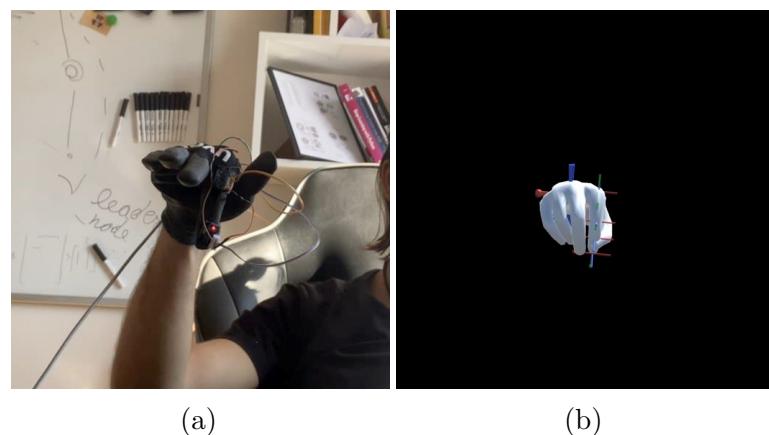
Figura 23



(a)

(b)

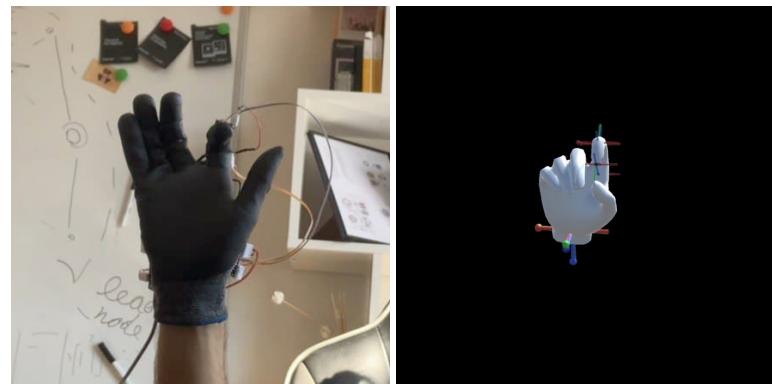
Figura 24



(a)

(b)

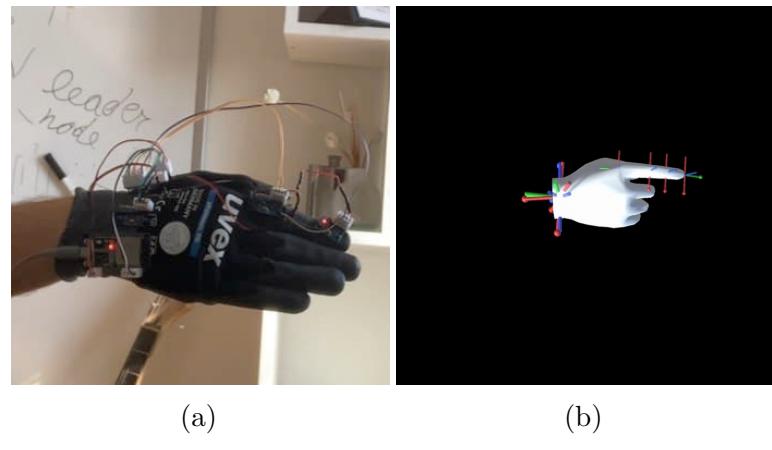
Figura 25



(a)

(b)

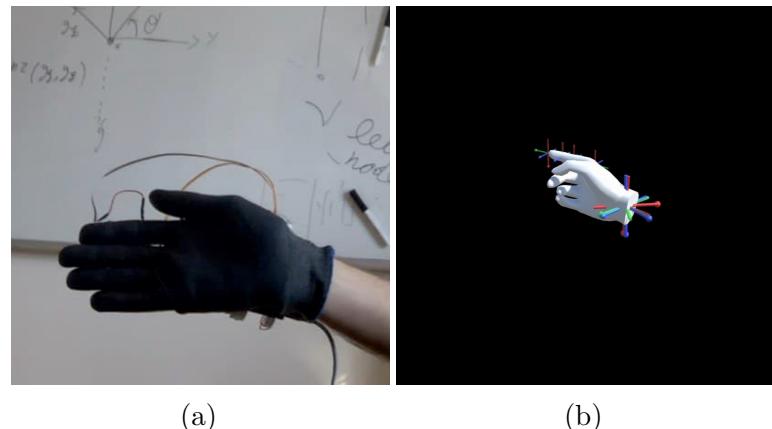
Figura 26



(a)

(b)

Figura 27



(a)

(b)

Figura 28



(a)

(b)

Figura 29



(a)

(b)

Figura 30

7 Conclusioni

Nella simulazione le dita sono state piegate a pugno così da poter visualizzare adeguatamente la posizione del dito.

Sono stati fatti dei test filmando la mano e la relativa animazione e catturandone dei frame in contemporanea così da poterle mettere a confronto.

Come possiamo vedere dalle figure la posizione della mano virtuale rispecchia quella della mano reale e anche la posizione del dito virtuale rispecchia adeguatamente quella del dito reale per ogni posizione della mano.

Riferimenti bibliografici

- [1] hideakitai. “MPU9250”. In: *GitHub* (2021). URL: <https://github.com/hideakitai/MPU9250>.
- [2] P.J. Kieliba et al. “Comparison of Three Hand Pose Reconstruction Algorithms Using Inertial and Magnetic Measurement Units”. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. 2018, pp. 1–9. DOI: 10.1109/HUMANOIDS.2018.8624929.
- [3] Rahul Kottath et al. “Multiple Model Adaptive Complementary Filter for Attitude Estimation”. In: *Aerospace Science and Technology* 69 (2017), pp. 574–581. ISSN: 1270-9638. DOI: <https://doi.org/10.1016/j.ast.2017.07.011>. URL: <https://www.sciencedirect.com/science/article/pii/S1270963816303418>.
- [4] Mayitzin. “Complementary Filter”. In: *Read the Docs* (2020). URL: <https://ahrs.readthedocs.io/en/latest/filters/complementary.html>.
- [5] Talat Ozyagcilar. “Calibrating an ecompass in the presence of hard and soft-iron interference”. In: *Freescale Semiconductor Ltd* (2012), pp. 1–17.
- [6] Talat Ozyagcilar. “Implementing a tilt-compensated eCompass using accelerometer and magnetometer sensors”. In: *Freescale semiconductor, AN 4248* (2012).
- [7] tockn. “MPU6050_tockn”. In: *GitHub* (2019). URL: https://github.com/tockn/MPU6050_tockn.