UNIVERSITÀ DEGLI STUDI DI PADOVA

—

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

—

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCANICA

PIATTAFORMA PER TELEOPERAZIONE DI MANIPOLATORE INDUSTRIALE COLLABORATIVO

RELATORE: CH.MO PROF. ING. GIULIO ROSATI

CORRELATORE: CH.MO ING. MATTEO BOTTIN

LAUREANDO: MATTEO TOSO

ANNO ACCADEMICO 2019-2020

Indice

Sommario

In questa tesi viene affrontato il tema della telerobotica, con l'obbiettivo di creare una piattaforma per teleoperazioni usando un robot collaborativo: il TechMan TM5-900.

Dopo una breve sintesi di quello che è un robot industriale, la storia e lo stato del mercato attuale, ci si concentra più sulla robotica collaborativa, descrivendo il modello utilizzato e la sua cinematica diretta e inversa.

Viene spiegato poi come il TM5 comunica tramite protocollo TCP-IP e di come è stato interfacciato con la piattaforma Matlab.

Viene descritta la classe di funzioni TM che muovono e comandano il robot tramite Matlab.

Viene riportata una breve analisi statistica sul comportamento del robot durante il movimento PTP, andando ad analizzare la traiettoria percorsa, precisione e ripetibilità dei movimenti.

Infine viene presentata una piattaforma per teleoperazioni tramite Joypad Xbox One.

Introduzione

L'utilizzo del termine Robot industriale come viene inteso oggi risale ai primi anni '60, quando nelle industrie sono comparsi i primi processi CAD e CAM, in accordo con la filosofia dell'epoca di automatizzare i processi. Il primo robot industriale è riconosciuto essere il braccio programmabile chiamato UNIMATE, utilizzato in processi potenzialmente pericolosi dalla General Motors.

Figura 1: UNIMATE, [1]

Il grande passo successivo è stato l'introduzione della tipologia SCARA nel 1978,un robot a 4 assi ideale per le applicazioni di pick & place.

Figura 2: SCARA, [2]

Assieme allo SCARA si sviluppavano parallelamente i robot antropomorfi, con 6 o più gradi di libertà, come ad esempio il robot PUMA, sviluppato nel 1978 da Unimation, la stessa dello UNIMATE.

Figura 3: PUMA, [3]

Col tempo i robot si sono evoluti a seconda delle necessita di mercato:i robot paralleli sono molto veloci e ottimi per le applicazioni pick & place,i multi-braccio

nel caso sia necessario avere più manipolatori e nel caso sia richiesto di ottimizzare la traiettoria oppure di muoversi in uno spazio di lavoro ridotto sono stati sviluppati i robot ridondanti a 7 assi.
Il mercato del robot è in continua espansione: nel 2018 il numero di nuovi installazioni è cresciuto del 6% rispetto all'anno precedente, fino a più di 420'000 nuove unità. L'andamento è in regolare crescita e si prevede che per gli anni successi si continui con questo trend.

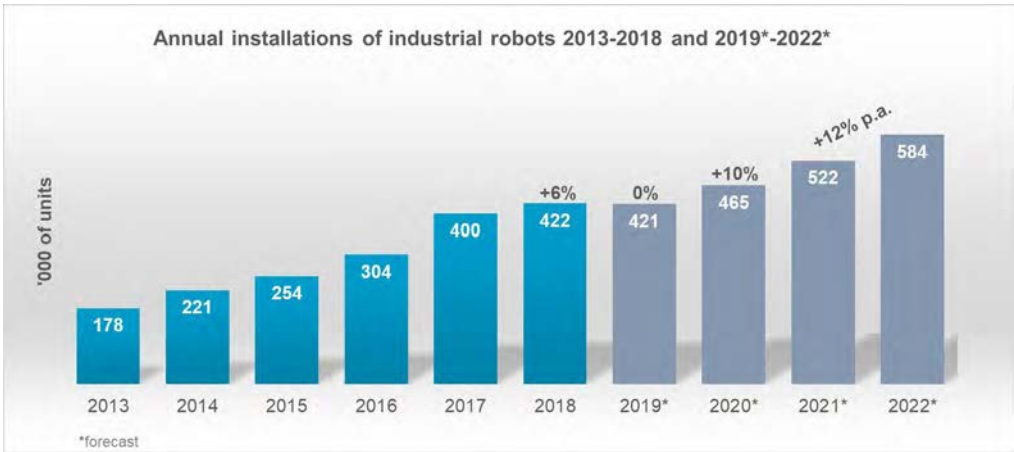


Figura 4: Numero di robot industriali installati e previsione per gli anni futuri, [4]

L'utilizzo dei robot si divide per il 30 % automotive, 25/30 % per elettronica, machinery per il 10%, chimica per il 5% e alimentare per il 3%, il restante 19% sono applicazione varie l'una dall'altra.

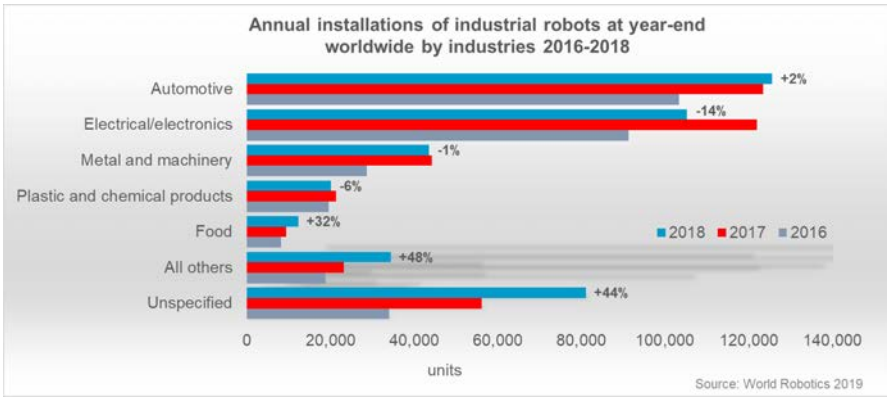


Figura 5: Divisione dei settori del mercato per i robot negli ultimi anni, [4]

Recentemente è comparsa nel mercato una nuova categoria: i robot collaborativi. Questi nascono con l'idea che l'uomo e la macchina possano cooperare in sinergia nello stesso spazio operativo. L'idea è nata nel 1996 da J. Edward Colgate e Michael Peshkin, professori alla Northwestern University. In pochi anni le maggiori case di produzione assieme ad altre nuove aziende nate esclusivamente per questa tipologia sono comparsi diversi modelli. Tra le prime aziende troviamo i KUKA, gli Universal Robot, i robot FANUC e la ABB.

In questa tesi verrà utilizzato un Robot Techman TM5-900

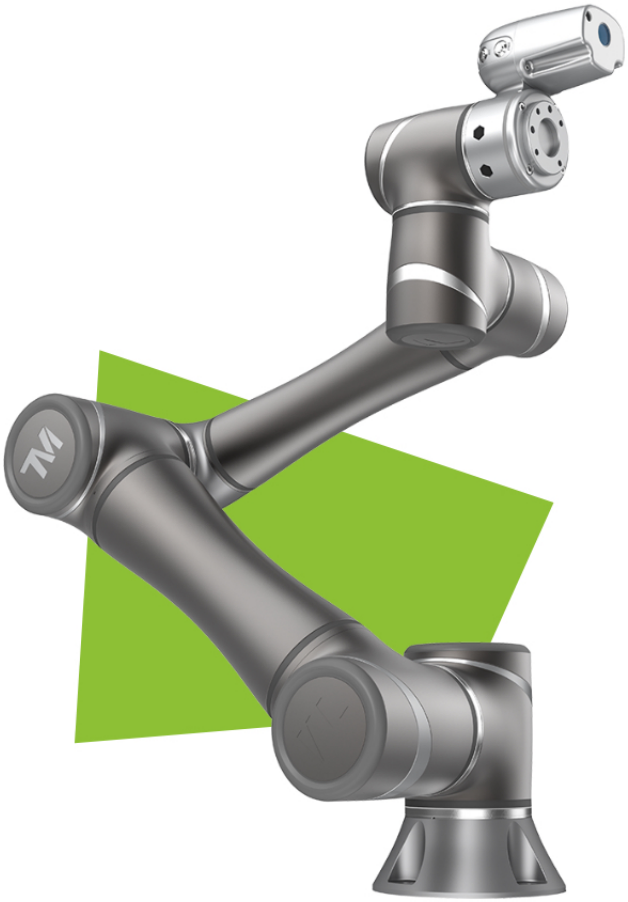


Figura 6: Robot Collaborativo TechMan TM5-900

Capitolo 1

TechMan TM-900

Sono state usate le fonti [5] [6] [7]

In questo capitolo verrà approfondito maggiormente il concetto di robotica collaborativa, dei suoi sviluppi attuali e futuri. Successivamente ci si concentrerà su quello che è il modello utilizzato, ovvero il Techman TM-900, descrivendo a sua struttura e la cinematica diretta ed inversa

1.1 Robotica Collaborativa

La differenza principale che distingue la robotica collaborativa da quella industriale tradizionale è proprio la stretta sinergia, che viene posta come obbiettivo, tra l'uomo e la macchina. Un robot industriale viene usato per automatizzare un'operazione su un prodotto, sostituendo completamente l'uomo. Per questo si usa in applicazioni semplici o ripetitive oppure in luoghi o condizioni che possono risultare pericolose. I robot industriali dunque non condividono lo spazio operativo con la forza umana, ma vengono chiusi in apposite celle in modo da impedirne uno scontro accidentale potenzialmente molto dannoso. Queste caratteristiche limitano i possibili campi di utilizzo ad applicazioni che possono essere completamente automatizzate o meno e ne limita la loro flessibilità , perché il robot può effettivamente essere riprogrammato per altre funzioni, ma il cambio comporterebbe ad una riprogettazione della cella, con i conseguenti costi in denaro e tempo.

Un cobot, diminutivo di robot collaborativo, nasce invece per lavorare assieme ad una persona: questi condividono lo stesso spazio di lavoro e vengono a contatto diretto l'uno con l'altro durante il ciclo produttivo. Dunque, venendo eliminati i vincoli del robot industriale, il cobot può andare a gestire tutti quei compiti durante applicazioni parzialmente automatizzate, alleggerendo il carico di lavoro dell'operaio ad esempio togliendo lavori ripetitivi, pesanti oppure pericolosi. Sembra lavori esattamente come gli industriali tradizionali, ma un cobot viene pensato per applicazioni in cui la presenza umana risulti ancora necessaria in alcuni passaggi, togliendo gli altri che altrimenti andrebbero a gravare sulla persona. Un ulteriore punto di forza del cobot è la flessibilità : non essendo vincolato ad una stazione è possibile posizionarlo su un carrello mobile e utilizzare lo stesso robot su più applicazioni diverse a seconda di quella che è la necessità di giornata.

Attualmente gli impieghi più frequenti riguardano:

- Handling (Pick and Place, Material Handling, confezionamento e pallettizzazione e Machine Tending)
- Assemblaggio e disassemblaggio (Avvitamento di viti e bulloni)
- Saldatura
- Operazioni con fluidi (incollaggio e verniciatura)
- Asportazione di truciolo (Grinding, Milling, and Cutting)
- altri (ispezione, controllo qualità , Die-casting and Molding)

I robot collaborativi sono più usati nelle seguenti industrie:

- Automotive
- Elettronica
- Lavorazione dei metalli,Plastica e altri polimeri
- alimentari
- Healthcare

- altri

1.2 Modello TM5-900

1.2.1 Tipologia

Il modello TM5-900 è un robot collaborativo a 6 assi, sviluppato dall'azienda taiwanese Techman. Fa parte della linea TM5 e ha uno sbraccio di 900 mm. Analizzando la sua struttura in generale è molto simile al tipo di collaborativo fondato da UR, con la differenza che questo è fornito di serie di una videocamera direttamente integrata con il software.

Nel laboratorio universitario il TM5 è stato comunque posizionato all'interno di una stazione chiusa, per motivi di sicurezza. Nella porta di ingresso è stato posizionato un sensore di apertura che è stato collegato al robot, di conseguenza con la porta aperta il robot non può entrare in funzione. Questo però è disattivabile nel caso si debba operare con free-move, ovvero una configurazione caratteristica dei cobot grazie alla quale tenendo premuto un tasto è possibile trascinare manualmente il robot nella posizione desiderata.

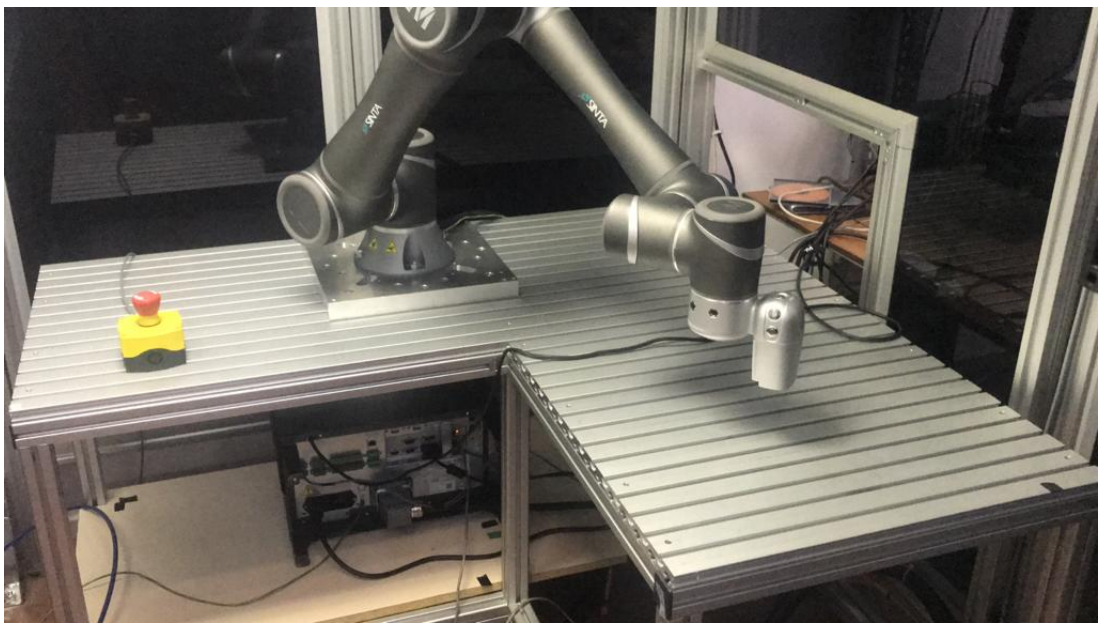


Figura 1.1: Postazione TM5 nel laboratorio dell'Università di Padova

1.2.2 Datasheet

Successivamente viene riportato il datasheet del robot. Da notare come le caratteristiche di velocità e ripetibilità siano peggiori rispetto a quelle di un robot industriale tradizionale, ma questo è comprensibile ricordando che le due tipologie di robot sono distinte per quanto riguarda le applicazioni e quando si valuta un cobot è più utile guardare come questo interagisca con una persona.

Collaborative Robots
TM5

Collaborative robot for assembly, packaging, inspection and logistics

- Built-in vision-based robot control enables visual servoing, inspection, and measurement.
- TMvision and landmark allows truly flexible and fast changeover.
- Easy-to-use graphical programming environment for quick startup.
- Plug & Play ecosystem makes it versatile for many manufacturing needs.
- Designed to promote a safer workplace with harmony between humans and machines.
- Integration with mobile robots provides fully autonomous logistics handling for a conveyor-less system.

TM5-700 **TM5-900**

Specification

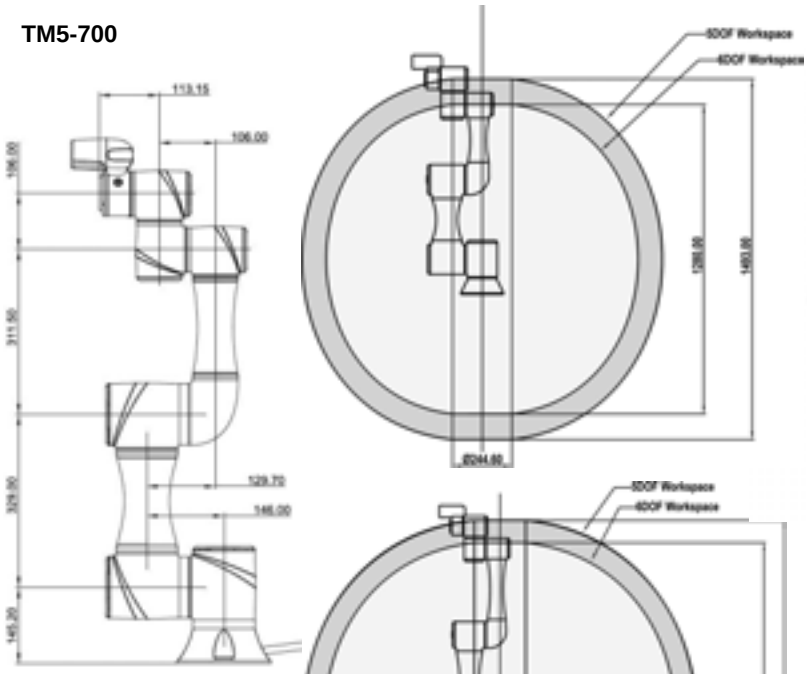
		TM5					
Product Name		**TM5-700**	**TM5M-700**	**TM5M-700 SEMI**	**TM5-900**	**TM5M-900**	**TM5M-900 SEMI**
Model		**RT6-0007000**	**RT6-0107000**	**RT6-0107010**	**RT6-0009000**	**RT6-0109000**	**RT6-0109010**
Weight (kg)		22.1			22.6		
Max Payload (kg)		6			4		
Reach (mm)		700			900		
Typical Speed (m/s)		1.1			1.4		
Joint Range	**Joint 1**	±270°					
	Joint 2, 4, 5	±180°					
	Joint 3	±155°					
	Joint 6	±270°					
Joint Speeds	**Joint 1, 2, 3**	180°/s					
	Joint 4, 5, 6	225°/s					
Repeatability (mm)		±0.05					
IP		IP54 (robot arm), IP32 (control box)					
Operating Temperature (°C)		0 to 50					
Power Supply		100-240 VAC, 50-60 Hz	22-60 VDC	22-60 VDC	100-240 VAC, 50-60 Hz	22-60 VDC	22-60 VDC
I/O Ports	**Control Box**	Digital In: 16 Digital Out: 16 Analog In: 2 Analog Out: 1					
	Tool	Digital In: 4 Digital Out: 4 Analog In: 1 Analog Out: 0					
I/O Interface		3 X COM, 1 X HDMI, 3 X LAN, 4 X USB2.0, 2 X USB3.0					
Communication		RS232, Ethernet (master), Modbus TCP/RTU (master & slave)					
Integrated Camera		5M pixels, color (AOI tasks: 5M/1.2M; others:1.2M)					
I/O Power Supply		24V 1.5A (control box and tool)					
Programming Environment		TMflow, flowchart based					
SEMI S2 Certified		No	No	Yes	No	No	Yes

*Manufactured by Techman Robot Inc.

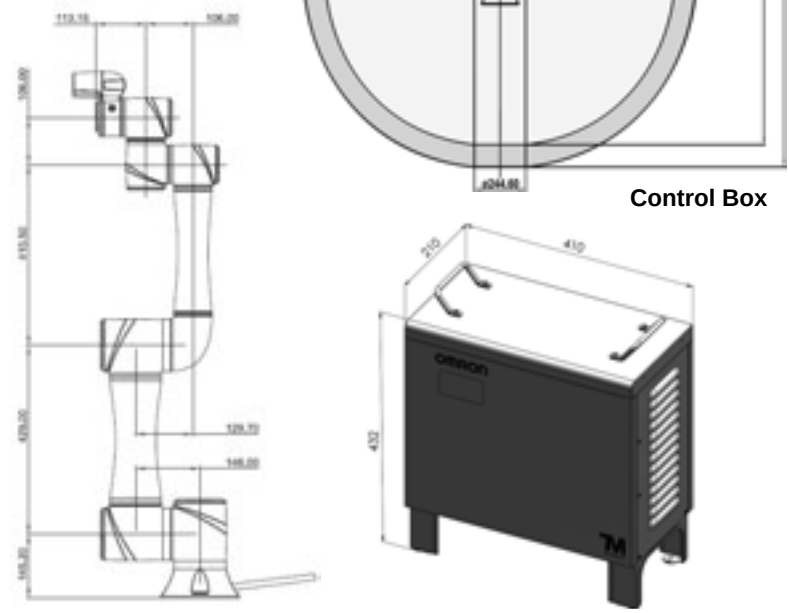
Dimensions

(Unit: mm)

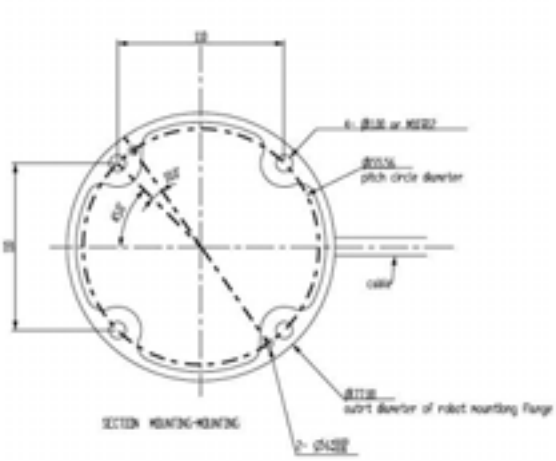
TM5-700



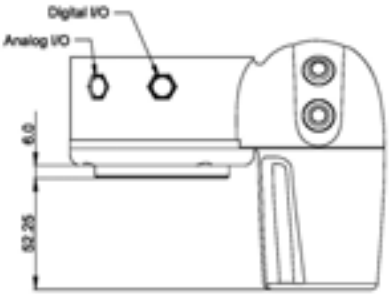
TM5-900



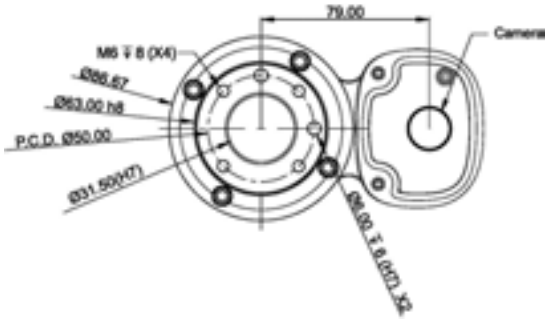
Footprint



Flange



Control Box



Robot Parts Code and Bundled Accessories

Type	TM5-700			TM5-900		
Product Name	**TM5-700**	**TM5M-700**	**TM5M-700 SEMI**	**TM5-900**	**TM5M-900**	**TM5M-900 SEMI**
Model	**RT6-0007000**	**RT6-0107000**	**RT6-0107010**	**RT6-0009000**	**RT6-0109000**	**RT6-0109010**
Overview	Robot with built-in vision system and control box			Robot with built-in vision system and control box		
Purpose	Typical for use in programming environment			Typical for use in programming environment		
Bundled Accessories	• TM5-700 robot arm (1) • Control box (1) • Calibration plates (one large and one small) • IO cables (2) • TM landmark (2) • Ground wire (1) • Power cable for the control box (4 standard model, 1 mobile model) • SEMI emergency OFF switch (SEMI model only)			• TM5-900 robot arm (1) • Control box (1) • Calibration plates (one large and one small) • IO cables (2) • TM landmark (2) • Ground wire (1) • Power cable for the control box (4 standard model, 1 mobile model) • SEMI emergency OFF switch (SEMI model only)		

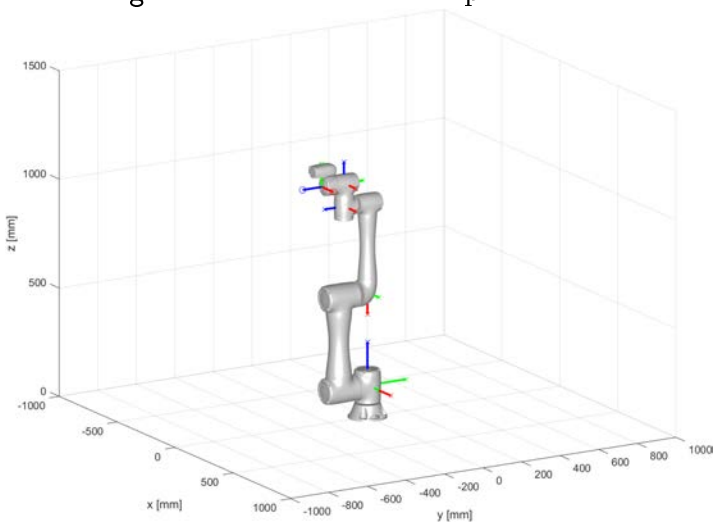
1.3 Cinematica

In questa sezione si analizza la cinematica diretta e indiretta del robot. L'ultima è stata poi tradotta in una funzione Matlab.

1.3.1 Cinematica diretta

Calcolare la cinematica diretta di un robot significa passare dalle coordinate di giunto alle coordinate dello spazio lavorativo. Partendo dalla Matrice di Denavit-Hartenberg si calcolano le matrici di trasformazione tra le varie terne: vista la struttura simile al UR, si utilizza la stessa configurazione per il TM5, con la differenza che la posizione non è sdraiata ma verticale e che per i giunti 2-3-4 le rotazioni positive coincidono con rotazioni orarie sull'asse Z.

Figura 1.2: Posizione Zero per il TM5



i	alpha i-1	ai-1	d	theta
1	0	0	145.1	0
2	90	0	0	-90
3	0	-429	0	0
4	0	-411.15	122.2	-90
5	90	0	106	0
6	-90	0	114.4	0

L'obbiettivo è dunque individuare la terna seguente:

$$ {}^{0}_{6}T\left(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6\right) = \begin{bmatrix} {}^{0}\hat{X}_{6x} & {}^{0}\hat{Y}_{6x} & {}^{0}\hat{Z}_{6x} & {}^{0}P_{6x} \\ {}^{0}\hat{X}_{6y} & {}^{0}\hat{Y}_{6y} & {}^{0}\hat{Z}_{6y} & {}^{0}P_{6y} \\ {}^{0}\hat{X}_{6z} & {}^{0}\hat{Y}_{6z} & {}^{0}\hat{Z}_{6z} & {}^{0}P_{6z} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

dove

$$ {}^{0}P_{6} = \begin{bmatrix} {}^{0}P_{6x} \\ {}^{0}P_{6y} \\ {}^{0}P_{6z} \end{bmatrix} $$

è l'origine della terna 6 vista dal sistema di riferimento 0

$$ {}^{0}\hat{Y}_{6} = \begin{bmatrix} {}^{0}\hat{Y}_{6x} \\ {}^{0}\hat{Y}_{6y} \\ {}^{0}\hat{Y}_{6z} \end{bmatrix} $$

è il versore di direzione y della terna 6 proiettato sulla terna 0

a questo punto si dive la matrice nelle sue trasformazioni di giunto in giunto:

$$ {}^{6}_{0}T\left(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6\right) = {}^{1}_{0}T(\theta_1){}^{2}_{1}T(\theta_2){}^{3}_{2}T(\theta_3){}^{4}_{3}T(\theta_4){}^{5}_{4}T(\theta_5){}^{6}_{5}T(\theta_6) $$

Conoscendo la formula della matrice elementare è possibile ricavare la forma completa e dunque risolvere il problema di cinematica diretta:

$$ {}^{i-1}_{i}T = \begin{bmatrix} \cos\theta_i & -sen\theta_i & 0 & a_{i-1} \\ sen\theta_i\cos(\alpha_{i-1}) & \cos\theta_i sen(\alpha_{i-1}) & -sen(\alpha_{i-1}) & -sen(\alpha_{i-1})d_i \\ sen\theta_i sen\left(\alpha_{i-1}\right) & \cos\theta_i sen(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

1.3.2 Cinematica inversa

Cinematica indiretta significa,partendo da un punto definito nello spazio operativo, ricavare un possibile valore degli angoli di giunto in modo tale che il robot si posizioni in quel punto. Il dato in ingresso è la trasformazione T_6^0, la soluzione gli angoli $[(\theta_1,...,\theta_6) \in [0; 2\pi[]$

Theta 1 Il primo angolo che si trova è θ_1,, per questo è sufficiente trovare la posizione della terna 5 rispetto alla terna di base, dunque $[^0P_5]$, e si può facilmente trovare arretrando di una distanza d_6 lungo l'asse z della terna 6, in modo da trovare il centro della terna 5. Si ricorda che entrambi T_6^0 e d_6 sono conosciuti.

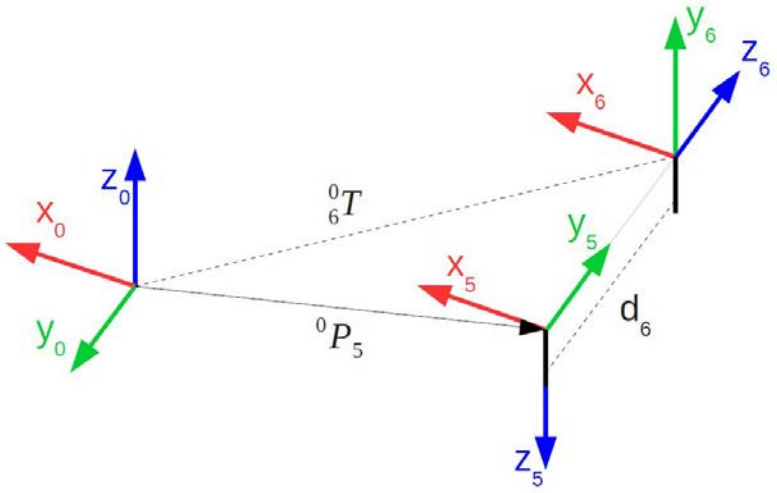


Figura 1.3: Trasformazione tra le terne 1-5-6

La traslazione può essere scritta come:

$$^0P_5 = {}^0P_6 - d_6 * {}^0\hat{Z}_6$$

$$^0P_5 = {}^0_6T * \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 1 \end{bmatrix}$$

Per andare a ricavare θ_1, si considera il centro del polso P_5: visto dalla terna zero e dalla terna 1, la differenza della sua posizione deve essere rotazione da zero a 1:

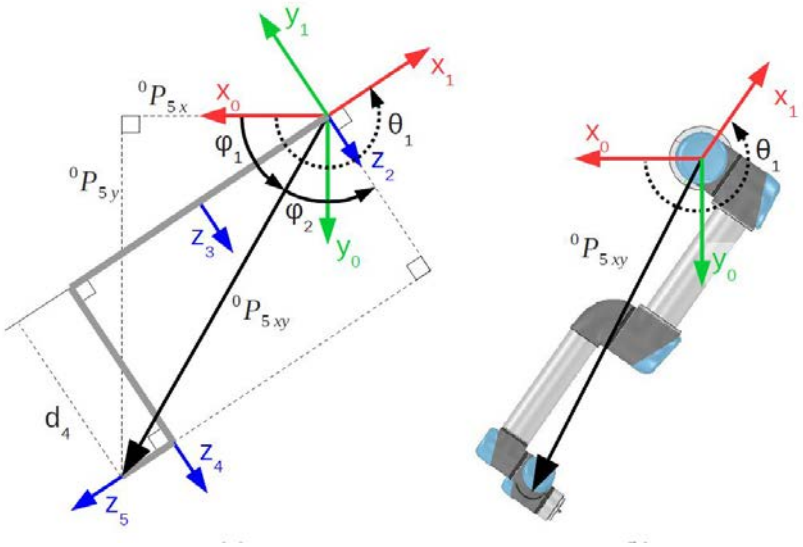


Figura 1.4: Robot, fino alla terna 5, visto dall'alto

$$v_{0\to1} = v_{0\to5} - v_{1\to5}$$
$$v_{0\to1} = v_{0\to5} + v_{5\to1}$$
$$\theta_1 = \phi_1 + (\phi_2 + \pi)$$

L'angolo ϕ_1 si ricava dal triangolo con lati P_5x^0 eP_5y^0:

$$\phi_1 = \arctan({}^0P_5y, {}^0P_5x)$$

L'angolo ϕ_2 invece si trova esaminando il triangolo a destra con ϕ_2 uno degli angoli: i due lati hanno valore di $|{}^0P_5xy|$ e d_4:

$$\cos(\phi_2) = \frac{d_4}{|{}^0P_{5xy}|}$$
$$\phi_2 = \pm\arccos(\frac{d_4}{|{}^0P_{5xy}|})$$
$$\phi_2 = \pm\arccos(\frac{d_4}{\sqrt{{}^0P_{5x}^2 + {}^0P_{5y}^2}})$$

dove il $\pm$ davanti all'arcoseno indica due possibili diverse configurazioni che può assumere il robot: braccio destro e braccio sinistro.

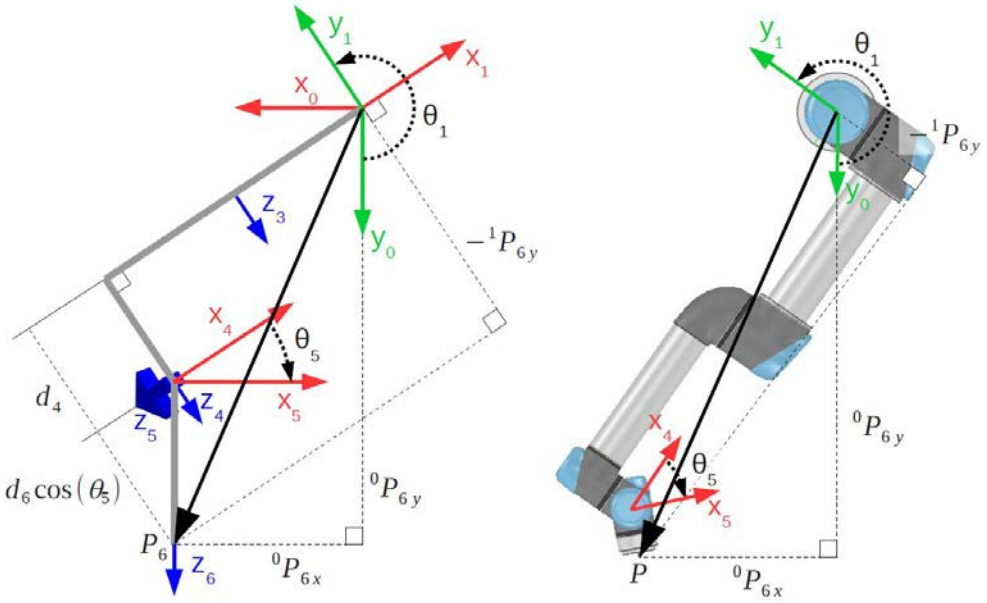


Figura 1.5: Robot completo visto dall'alto

Theta 5 Per trovare θ_5 si guarda ancora il robot dall'alto: si nota che ${}^1P_{6y}$, la componente in y del centro della terna 5 vista da 1, è solo in funzione di θ_5

dalla figura si vede come:

$$-{}^1P_{6y} = d_4 - d_6 \cos\theta_5$$

è possibile esprimere ${}^1P_{6y}$ come il risultato di una rotazione di ${}^0P_6$ attorno z_1

$$
\begin{aligned}
{}^0P_6 &= {}^0_1R * {}^1P_6 \\
{}^1P_6 &= {}^0_1R^T * {}^0P_6 \\
\begin{bmatrix} {}^1P_{6x} \\ {}^1P_{6y} \\ {}^1P_{6z} \end{bmatrix} &= \begin{bmatrix} \cos\theta_1 & -sen\theta_1 & 0 \\ sen\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \begin{bmatrix} {}^0P_{6x} \\ {}^0P_{6y} \\ {}^0P_{6z} \end{bmatrix} \\
\begin{bmatrix} {}^1P_{6x} \\ {}^1P_{6y} \\ {}^1P_{6z} \end{bmatrix} &= \begin{bmatrix} \cos\theta_1 & sen\theta_1 & 0 \\ -sen\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^0P_{6x} \\ {}^0P_{6y} \\ {}^0P_{6z} \end{bmatrix} \\
{}^1P_{6y} &= {}^0P_{6x}(-\sin\theta_1) + {}^0P_{6y}\cos\theta_1
\end{aligned}
$$

combinando le equazioni precedenti otteniamo l'espressione di θ_5

$$-d_4 - d_6 * \cos\theta_5 = {}^0P_{6x}(-\sin\theta_1) + {}^0P_{6y}\cos\theta_1$$
$$\cos\theta_5 = \frac{{}^0P_{6x}\sin\theta_1 - {}^0P_{6y}\cos\theta_1 - d_4}{d_6}$$
$$\theta_5 = \pm\arccos(\frac{{}^0P_{6x}\sin\theta_1 - {}^0P_{6y}\cos\theta_1 - d_4}{d_6})$$

anche in questo caso sono è presente un $\pm$, sono le due possibili soluzioni di polso flippato e non flippato. Da notare che le soluzioni sono possibili solo nel caso in cui $|{}^1P_{6y} - d_4| \leq |d_6|$.

Theta 6 Per trovare θ_6 si va ad esaminare y_1 visto dalla terna 6;${}^6\hat{Y}_1$. Questo asse sarà sempre parallelo agli assi ${}^6\hat{Z}_{2,3,4}$, dunque è in funzione solo di θ_5 e θ_6.

Essendo in funzione di due angoli, è possibile eseguire un cambio di coordinate e passare a coordinate sferiche, dove $-\theta_6$ è l'azimuth e la coordinata polare è θ_5.

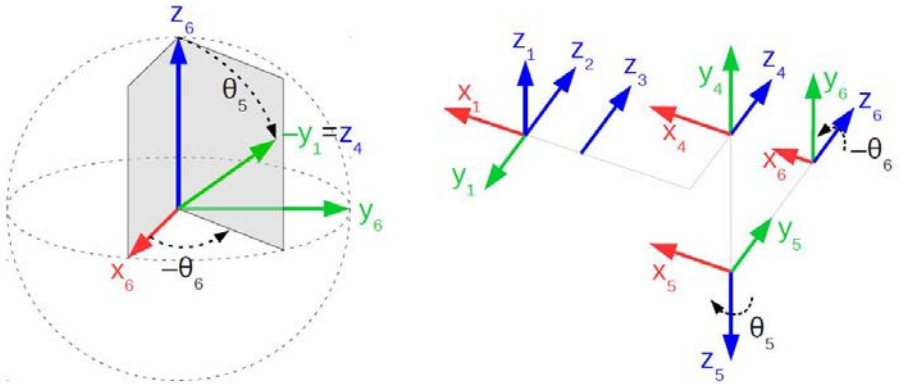


Figura 1.6: l'asse -${}^6\hat{Y}_1$ espresso in coordinate polari

Convertendo dunque ${}^6\hat{Y}_1$ in coordinate sferiche si ottiene:

$$-{}^6\hat{Y}_1 = \begin{bmatrix} \sin\theta_5\cos(-\theta_6) \\ \sin\theta_5\sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix}$$
$$ {}^6\hat{Y}_1 = \begin{bmatrix} -\sin\theta_5\cos(\theta_6) \\ \sin\theta_5\sin(\theta_6) \\ -\cos\theta_5 \end{bmatrix}$$

Nell'ultima equazione si nota come sia possibile isolare l'anglo θ_6 in funzione delle componenti di ${}^6\hat{Y}_1$, ma l'obbiettivo è avere una dipendenza diretta da ${}^6_0T$,

per questo identifichiamo ${}^6\hat{Y}_1$ come risultato di una rotazione di θ_1 del piano x-y della terna 0:

$$
\begin{aligned}
{}^6\hat{Y}_1 &= {}^6\hat{X}_0 * (-sen\theta_1) + {}^6\hat{Y}_0 * \cos\theta_1 \\
{}^6\hat{Y}_1 &= \begin{bmatrix} -{}^6\hat{X}_{0x} * sen\theta_1 + {}^6\hat{Y}_{0x} * \cos\theta_1 \\ -{}^6\hat{X}_{0y} * sen\theta_1 + {}^6\hat{Y}_{0y} * \cos\theta_1 \\ -{}^6\hat{X}_{0y} * sen\theta_1 + {}^6\hat{Y}_{0y} * \cos\theta_1 \end{bmatrix}
\end{aligned}
$$

Eguagliando le prime due equazioni di quest'ultima serie con le prime due della serie precedente otteniamo:

$$
\left\{ \begin{array}{l} -\sin\theta_5 \cos(\theta_6) = -{}^6\hat{X}_{0x} * sen\theta_1 + {}^6\hat{Y}_{0x} * \cos\theta_1 \\ \sin\theta_5 \sin(\theta_6) = -{}^6\hat{X}_{0y} * sen\theta_1 + {}^6\hat{Y}_{0y} * \cos\theta_1 \end{array} \right\}
$$

$$
\left\{ \begin{array}{l} \cos(\theta_6) = \frac{{}^6\hat{X}_{0x} * sen\theta_1 - {}^6\hat{Y}_{0x} * \cos\theta_1}{\sin\theta_5} \\ \sin(\theta_6) = \frac{-{}^6\hat{X}_{0y} * sen\theta_1 + {}^6\hat{Y}_{0y} * \cos\theta_1}{\sin\theta_5} \end{array} \right\}
$$

Da cui poi ricaviamo direttamente la formula:

$$
\theta_6 = \arctan 2(\frac{-{}^6\hat{X}_{0y} * sen\theta_1 + {}^6\hat{Y}_{0y} * \cos\theta_1}{\sin\theta_5}, \frac{{}^6\hat{X}_{0x} * sen\theta_1 - {}^6\hat{Y}_{0x} * \cos\theta_1}{\sin\theta_5})
$$

La soluzione in caso θ_5 sia uguale a zero è indeterminata, questo perché i giunti 2,3,4 e 6 risulterebbero allineati, sono troppi gradi di libertà per una soluzione univoca.

Theta 3 I rimanenti giunti 2,3,4 hanno gli assi paralleli tra di loro, dunque sono assimilabili ad un manipolatore a 3 assi sul piano.

Avendo a disposizione gli angoli 1,5,6 possiamo ricavare la terna ${}^1_4T$, terna 4 vista dalla terna 1, Dalla figura si vede come la traslazione$|{}^1P_{4xz}|$ sia dipendente solo da θ_3

si ricava dunque che l'angolo θ_3 sia:

$$
\begin{aligned}
\cos\phi_3 &= \frac{(-a_2)^2 + (-a_3)^2 - \left|{}^1P_{4xz}\right|^2}{2(-a_2)(-a_3)} = \frac{a_2^2 + a_3^2 - \left|{}^1P_{4xz}\right|^2}{2a_2a_3} \\
\cos\theta_3 &= \cos(\pi - \phi_3) = -\cos\phi_3 \\
\cos\theta_3 &= -\frac{a_2^2 + a_3^2 - \left|{}^1P_{4xz}\right|^2}{2a_2a_3} \\
\theta_3 &= \pm \arccos(-\frac{a_2^2 + a_3^2 - \left|{}^1P_{4xz}\right|^2}{2a_2a_3})
\end{aligned}
$$

la soluzione esiste solo se l'argomento dell'arcoseno è compreso tra $[-1; 1]$,ovvero $|{}^1P_{4xz}| \in [|a_2 - a_3|\,; |a_2 + a_3|]$ Anche qui si trovano due soluzioni possibili: queste corrispondono alla configurazione di gomito alto e gomito basso.

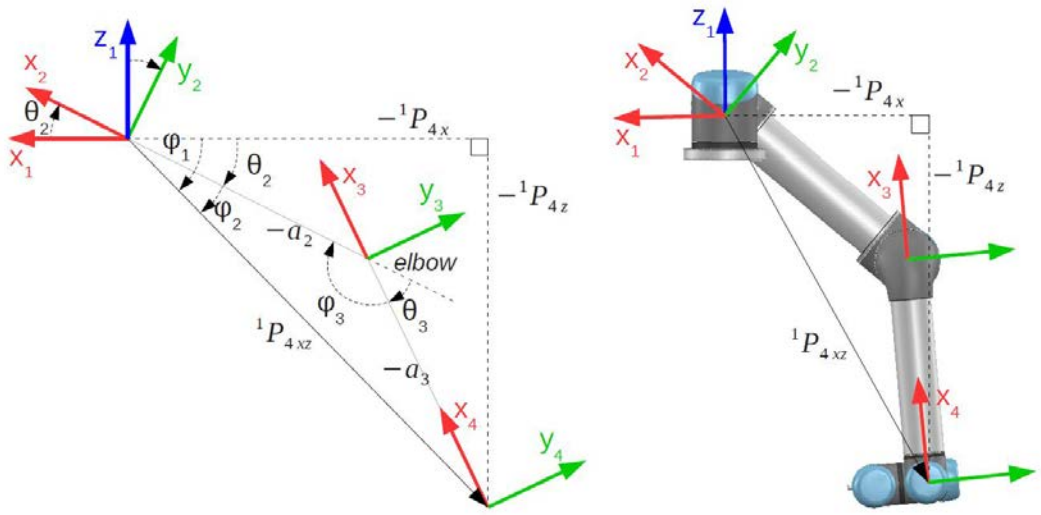


Figura 1.7: I giunti 2,3,4 assieme formano un manipolatore planare

Theta 2 L'angolo θ_2 può essere individuato come $\phi_1 - \phi_2$:

$$\phi_1 = \arctan 2(-{}^1P_{4z}, -{}^1P_{4x})$$
$$\phi_2 = arcsen(\tfrac{-a_3 * sen(\theta_3)}{|{}^1P_{4xz}|})$$
$$\theta_2 = \arctan 2(-{}^1P_{4z}, -{}^1P_{4x}) - arcsen(\tfrac{-a_3 * sen(\theta_3)}{|{}^1P_{4xz}|})$$

Theta 4 L'ultimo angolo θ_4 si ricava da come angolo tra X_3 e X_4 lungo Z_4, può essere dunque facilmente espresso dall'ultima matrice di trasformazione mancante ${}^3_4T$ dalla prima colonna ${}^3\hat{X}_4$:

$$\theta_4 = \arctan 2({}^3\hat{X}_{4y}, {}^3\hat{X}_{4x})$$

Si sono dunque trovate tutte le soluzioni agli angoli dei giunti, alla fine ci sono 8 configurazioni possibili per il TM5 a seconda di braccio, gomito e polso.

Una volta trovati gli angoli dei giunti questi vanno corretti con quello che è l'angolo di offset della posizione zero, in questo caso:$qoff = [0, -90, 0, 90, 0, 0]$ Infine si correggono gli angoli con quello che è il loro verso positivo di rotazione, in questo caso $qsign = [1, -1, -1, -1, 1, 1]$.

Capitolo 2

Interfacciamento con Matlab

Sono state usate le fonti [7] [8] [9] [10] [7]

In questo capitolo ci si occupa della connessione tra Robot e Matlab:come si controlla il TM tramite TMFlow, come il robot riceve le informazioni tramite tcp-ip, l'impostazione di Matlab in modo da preporlo alla comunicazione in rete, come le informazioni devono essere strutturate e alcuni esempi di controllo.

2.1 Comandare il Robot

Il TM ha un suo software con il quale viene controllato: TMFlow. Il nome sottolinea proprio come l'utente non sia obbligato ad usare un vero e proprio linguaggio di programmazione ma si utilizzi una struttura simile a un diagramma di flusso nel quale ogni blocco è un azione diversa, dunque anche senza imparare un nuovo linguaggio è possibile mettere subito il TM al lavoro.
L'obbiettivo di questa tesi però è di riuscire a interfacciare il robot TM5 tramite l'applicazione matlab, sfruttando le funzioni già create precedentemente, creando così una piattaforma per teleoperazioni.

Il primo passo riguarda la costruzione di un progetto TMFlow contenente il nodo Listen: questo fa entrare il robot nello stato di listen, ovvero il robot si ferma si aspetta comandi tramite tcp ip, che poi esegue. Vediamo dunque il progetto utilizzato.

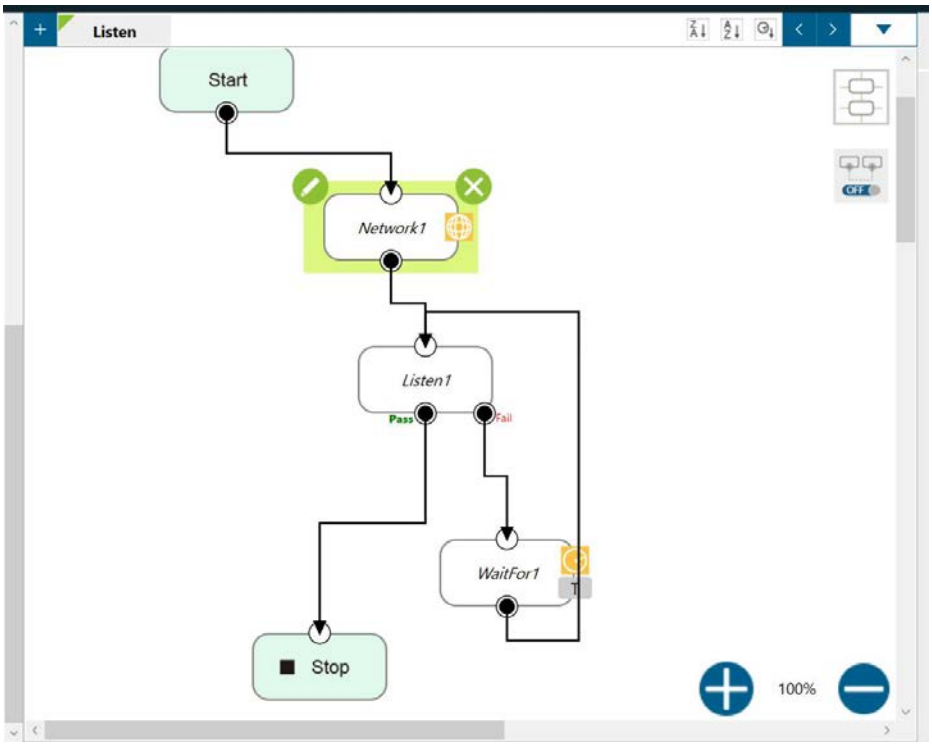


Figura 2.1: Schema di flusso del progetto Listen1

Lo schema di flusso è molto semplice, in quanto sono presenti solo 3 nodi, più inizio e fine.

Partendo dal nodo Start, il flusso raggiunge il nodo network: qui viene definito il Device "MioPC" comprendente indirizzo IP e porta del PC che attualmente viene usato per comandare il robot, in questo caso $ip =$ "169.254.237.104 e porta 5895, in teoria questo passaggio non dovrebbe essere necessario, tuttavia si rileva che se non registrato tra i device disponibili i passaggi successivi attraverso il nodo Listen non vengono eseguiti correttamente dalla macchina. Il resto delle impostazioni è superfluo.

Successivamente si entra nel nodo Listen, nel quale si può decidere un messaggio personalizzato da mandare una volta avvenuto il collegamento, in questo caso " entrato in Listen1" e le impostazioni di connection timeout data timeout. Consigliato invece spuntare la casella di "Print received log data", cosicché si possa vedere nel log lo scambio di pacchetti in tempo reale, utile in caso di errore.

Si entra quindi nel nodo wait, ma il valore di temposarà nullo dunque si torna in un loop nel nodo listen.

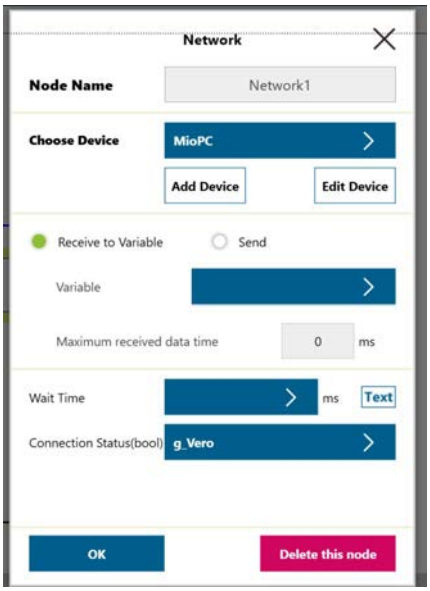


Figura 2.2: Impostazione nodo network

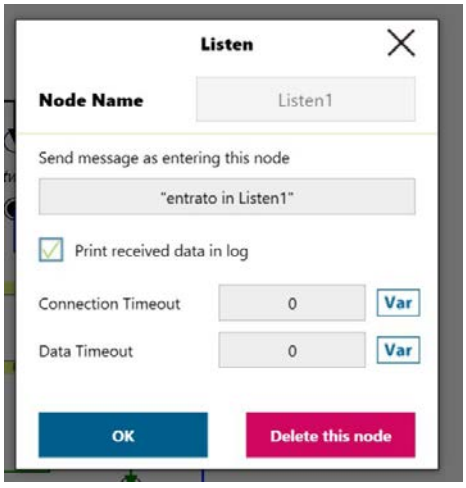


Figura 2.3: Impostazioni nodo listen

Il nodo Stop si raggiunge solo se dal nodo listen ne viene dato il comando.

Lo schema del flusso è dunque finito, avviandolo tramite il terminale del robot (il telecomando), il programma viene eseguito in modalità manuale al 5% della velocità massima. Per aumentare la velocità gradualmente si utilizzano i pulsanti + e - nel terminale, con incrementi del 5%. Confermata quindi la velocità, sempre con programma eseguito in modalità manuale, si preme il pulsante M/A nel terminale, così il progetto viene salvato con la sua velocità in modalità automatica e non è più necessario avviarlo tramite TMFlow, ma semplicemente premendo il pulsante play/pause nel terminale.

Finito lo schema del flusso è più comodo andare nelle impostazioni del robot e settare un indirizzo ip fisso, così i codici matlab non devono essere modificati ad ogni accesso. L'indirizzo ip del robot utilizzato è "169.254.138.9"

Questi sono i passaggi lato TMflow che è necessario per interfacciare il TM5 tramite matlab, ma una volta conclusi TMFlow non sarà più necessario.

2.2 Strutturare la connessione

Inizia dunque il lavoro su matlab vero e proprio: innanzitutto è necessario scaricare il pacchetto Instrument control toolbox da Mathworks, in modo tale da avere più funzioni per gestire meglio la comunicazione tramite TCP-IP.

Si crea uno script matlab per creare la connessione tra PC e robot: il nome del file in questo caso è ClientListen.m, riportato in seguito

Listing 2.1: File ClientListen.m per stabilire una connessione con il TM5

```
1
2  clear
3  clc
4  %% Connessione
5  %istruzioni
6  % Avviare un script singolo con questa funzione, entro pochi secondi
7  % avviare il Progetto del robot ( dovrebbe essere in auto, luce blu)
8  % tramite il tasto play del telecomando, attendere la risposta.
9
10 %funzione
11
12 % connetto a TM5, tramite il suo ip e la porta 5890, definisco il network role di
13 % questo PC come client
```

```
14 connessione = tcpip('169.254.138.9', 5890, 'NetworkRole', 'client');
15 %definisco il terminator dei pacchetti
16 connessione.Terminator='CR/LF';
17 connessione.InputBufferSize=512;
18 connessione.OutputBufferSize=512;
19 %apro la connessione
20 fopen(connessione);
21 %pausa per dare tempo al robot di rispondermi positivamente
22 pause(0.01)
23 %leggo la risposta sull'ingresso del nodo Listen, qui compare l'intero pacc
24 fscanf(connessione,'%s')
```

Il codice è ben commentato in modo tale che chiunque lo debba utilizzare in futuro non abbia problemi.

Le prime linee di comando cancellano variabili esistenti e puliscono la command window. Il commento alle prime righe spiega la successione di eventi in modo da collegare il robot. Si avvia con il tasto play sul terminale del TM5 il progetto Listen già pre-impostato come automatico, entro pochi secondi si attiva questo script e aspettando un attimo se tutto va bene compare nella command window il pacchetto di risposta contenente il messaggio personalizzato che è stato inserito nel nodo listen.

Tramite la funzione *tcpip* si crea un oggetto *connessione* che rappresenta il metodo di comunicazione tra pc e TM5. i campi sono:

- indirizzo ip del robot: " 169.254.138.9 "
- porta di comunicazione:" 5890 " come specificato da manuale
- ruolo del computer nella connessione, ovvero client: si specifica "'NetworkRole','Client'"

così si crea la connessione, ora vengono definiti alcuni suoi parametri:

- Terminator: il terminator è l stringa di byte che chiude il pacchetto inviato, si è scelto il tipo CR/LF
- BufferSize in input e output: per entrambi valore 512

La connessione adesso è pronta, dunque tramite la funzione $fopen(connessione)$ viene aperta, si attende per 0.01 secondi la risposta del robot e infine tramite la funzione $fscanf(connessione,'\%s')$ la si visualizza in command window.

2.3 Pacchetto TCP-IP

Per comunicare tramite tcp-ip il TM5 chiede che gli si venga inviato un pacchetto di dati secondo la sua configurazione:

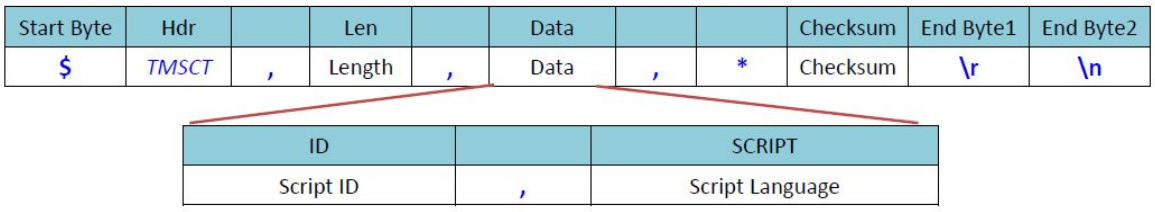


Figura 2.4: Struttura pacchetto come descritto dal manuale

Start Byte è il Byte iniziale di tutto il pacchetto, quando al nodo listen arriva una informazione che inizia con questo byte, si aspetta un pacchetto. quando il robot manda una risposta, questa inizia con questo byte

Ha dimensione 1 (è un solo byte), in ascii è rappresentato dal simbolo del dollaro $ e in linguaggio esadecimale da 0x24.

Header è la stringa che defisince la funzione del comando nel pacchetto

- TMSCT introduce uno script esterno, quasi sempre verrà usato questo
- TMSTA introduce una richiesta di informazioni da parte del PC nei confronti del TM, ad esempio quando si vuole chiedere la posizione nello spazio dei giunti.
- CPERR è usato solamente dal TM per comunicare un errore e il suo tipo

Length Lunghezza dei dati come indicato nello schema precedente, può essere scritta in UTF8, decimali esadecimali o binario. Nel nostro caso significa il numero di caratteri al suo interno.

Data formato dall'id del comando: un indice da 0 a 99 per distinguerlo dagli altri comando e il comando vero e proprio.

Checksum è una coppia di byte calcolata dal pacchetto completo, questo serve a verificare che durante la comunicazione non ci sia stata perdita di informazioni dovuto a cause esterne: il checksum viene calcolato e inviato dal PC e poi ricalcolato dal robot, se corrisponde a quello arrivato significa che la comunicazione è avvenuta con successo.

Il TM5 utilizza XOR dal simbolo del dollaro all'asterisco esclusi.

EndByte Sancisce la fine del pacchetto destinato al robot, in questo caso sono due i pacchetti per questo compito:$\backslash r \backslash n$.

Tuttavia con questa configurazione di terminale i Robot Techman restituiscono errore. Andando a vedere la struttura del pacchetto inviato, si evidenzia come attorno al pacchetto che il TM riceve matlab costruisce il pacchetto completo che poi viene inviato, aggiungendo le informazioni come indirizzo ip destinatario, quale porta usare, un tag temporale e infine header e terminal.

Il terminal aggiunto da matlab si fondeva con quello scritto manualmente, il TM non era in grado di distinguere le due cose e ritornava errore nonostante fosse sintatticamente corretto.

Per evidenziare l'errore è stato utilizzato il software WireShark, in grado di individuare i singoli pacchetti scambiati all'interno di una rete. In figura si vede l'esempio di un pacchetto inviato dal robot e destinato al PC, con evidenziato la parte "data".

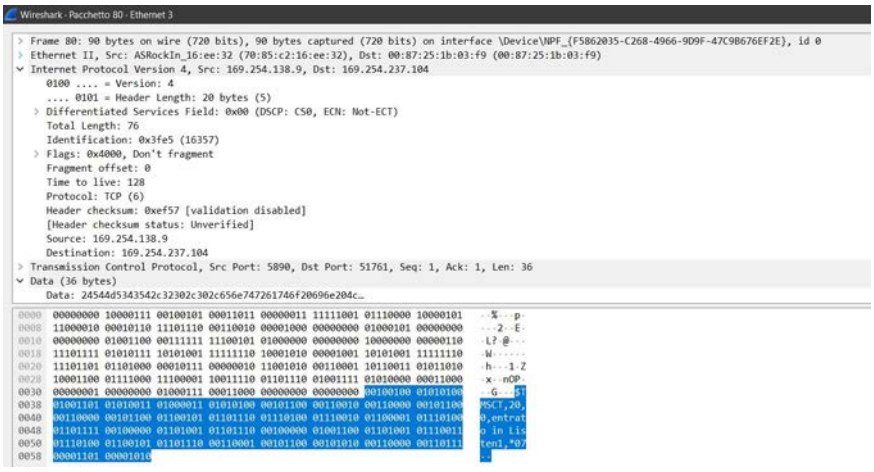


Figura 2.5: Schermata Wireshark con pacchetto ricevuto dal robot

Confrontando un pacchetto spedito da matlab, formalmente corretto ma effettivamente con errore, con un pacchetto spedito direttamente dal robot, si notava una differenza di come nel primo caso parte del terminal non veniva riconosciuto come tale e sforava nella parte data. Di conseguenza è sufficiente modificare il terminal scritto manualmente da $\backslash r \backslash n$ a $\backslash n$ In questa configurazione non ci sono errori e si può quindi comunicare con il robot.

2.4 Esempio Comunicazione

Listing 2.2: Esempio di file matlab con alcuni pacchetti completi di comunicazione

```
%Cambio Digital Output da 0 a 1
%'$TMSCT,26,1,IO["ControlBox"].DO[1]=1,*64\n'

%cosi accendo/spengo la luce
%$TMSCT,24,1,Robot[0].CameraLight=0,*5C\n'

%ricevo coordinate dal robot
%'$TMSCT,36,1,ListenSend(90,Robot[0].CoordRobot),*15\n'
```

Nel codice proposto sono presenti esempi di pacchetto, con la loro funzione.

Nel primo caso vogliamo impostare il valore di un Digital Output da 0 a 1, riconosciamo le strutture precedentemente elencate: Header, lunghezza, data , checksum e byte terminalem, da notare come il byte terminale sia non quello definito da manuale ma la sua versione funzionante. Nel secondo caso si accende e spegne la luce presente nel modulo della camera al giunto 6, Nel terzo caso si richiedono le coordinate del robot nello spazio dei giunti.

Capitolo 3

Classe TM

Sono state usate le fonti [9] [7] [10]

In questo capitolo verrà approfondita e descritta la classe TM di matlab, creata per poter interagire con il robot TM5. Si presenteranno prima la lista delle funzioni e poi una alla volta verranno riportate e descritte.

3.1 Funzioni

Listing 3.1: File TM.m contenente tutte le funzioni

```
classdef TM<handle
% classe per le funzioni di calcolo terne

    methods (Static)
        Ans = Connessione()
        q=ask(connessione,id);
        answer=PTP(connessione,txt,coord,vel , Ta,racc ,PrecPositioning, conf )
        answer=PLine(connessione,txt,coord,vel,Ta,racc)
        answer=Line(connessione,txt,coord,speed,Ta,racc,PrecPos)
        answer=Circle(connessione,txt,puntomezzo,puntofin,speed,Ta,racc,ArcAngle,PrecPos)
        answer = Move_PTP(connessione,txt, coord,vel,Ta,racc,PrecPos,conf)
        answer=Move_PLine(connessione,txt,coord,vel,Ta,racc)
        answer = Move_Line(connessione,txt,coord,vel,Ta,racc,PrecPos)
        I=ReadIO(connessione,fonte,tipo,id)
        answer=WriteIO(connessione,fonte,tipo,id,valore)
        I=InstantReadIO(connessione,fonte,tipo, id)
        answer=InstantWriteIO(connessione,fonte,tipo,id,valore)
    end
end
```

Oltre a queste funzioni sono presenti due funzioni accessorie che vengono usate dalle altre, una calcola il checksum e l'altra scrive il pacchetto completo.

3.2 Funzione checksum

La funzione checksum, come suggerisce il nome, parte dai dati e ne calcola il checksum.

Listing 3.2: Funzione checksum.m che restituisce il checksum come richiesto dal robot TM5

```
%% Funzione CheckSum
% il checksum è un insieme di due cifre esadecimali che, vengono inviate
% assieme al pacchetto, il robot ricalcola il checkum per conto suo e se
% quello ricalcolato è uguale a quello ricevuto significa che il pacchetto
% è arrivato tutto senza errori.

%function checksum = checksum(String)
% si usa uno XOR esadecimale
function cs = checksum(String)


String_d = double(String).';    % da char a double numerico, trasponendo
[N,M]=size(String_d); % misuro la dimensione della stringa, in modo da capire
                      % quante volte ripetere l'operazione XOR
cs = String_d(1,:);  %primo valore del CS, ovvero primo valore della String

for i = 2:N   %ripeto da 2 a N

            cs(1,:) = bitxor(cs(:),String_d(i,:));

end

    cs=dec2hex(cs);%ritrasformo in esadecimale

     if length(cs) == 1  % se il CS risulta ad una sola cifra, ci aggiungo uno zero davanti
        cs = strcat('0',cs);
     end
 end
```

Come primo passo si prende la stringa di comando in ingresso e la si converte interamente da string a double, cosi ogni carattere diventa un numero in double e si restituisce il vettore String_d i cui membri sono numeri corrispondenti a un carattere del pacchetto iniziale.

Successivamente viene misurata la lunghezza di questo vettore e si inizializza il checksum come il primo numero del vettore. Si esegue l'operazione XOR tra il checksum attuale ed il membro del vettore di indice successivo a quello con cui è stato calcolato il checksum attuale. Si procede con un ciclo for finché non si ottiene il checksum finale. Questo poi viene espresso in base esadecimale. Nel caso in cui sia composto da una sola cifra, ci si aggiunge 0 alla fine perché il CS come richiesto dal TM5 deve sempre avere 2 cifre.

3.3 Funzione writepack

La funzione writepack, come suggerisce il nome,scrive il pacchetto completo partendo dalle informazioni in ingresso, che sono header e comando.

Listing 3.3: Funzione writepack.m che scrive il pacchetto completo

```
%% Funzione writepack
% Preso il comando, scrive il pacchetto completo da spedire poi al Robot
function pack= writepack(head,comando)

%aggiungo Header, che è TMSTA O TMSCT  a seconda di quello che mi serve,
% quasi sempre TMSCT
%ottengo la stringa che poi viene analizzata per il checksum, da dollaro ad
%asterisco eslcusi
unchecked=char(strcat(head,',',length(char(comando)),',',comando,','));


%calcolo il checksum
cs=checksum(unchecked);

%completo il pacchetto
pack=strcat('$',unchecked,'*',cs,'\n');

end
```

Partendo dal comando in ingresso, si inizia a comporre il pacchetto. Come primo passo si compone tutta la parte di pacchetto che deve essere sottoposta alla composizione del checksum. Tramite la funzione strcat si uniscono header, lunghezza del comando, e comando stesso. da questa poi calcolo il checksum e infine unisco gli ultimi componenti per formare il pacchetto completo: \$, stringa unchecked, simbolo $*$, checksum e byte terminale $\backslash n$.

3.4 Funzione Connessione

La funzione connessione è un riassunto del file ClientListen.m già descritto prima.

Listing 3.4: Funzione Connessione.m che connette robot e PC

```
1
2  function connessione = Connessione()
3  %% Connessione
4  %istruzioni
5  % Avviare un script singolo con questa funzione, entro pochi secondi
6  % avviare il Progetto del robot ( dovrebbe essere in auto, luce blu)
7  % tramite il tasto play del telecomando, attendere la risposta.
8
9  %funzione
10
11 % connetto a TM5, tramite il suo ip e la porta 5890, definisco il network role di
12 % questo PC come client
13 connessione = tcpip('169.254.138.9', 5890, 'NetworkRole', 'client');
14 %definisco il terminator dei pacchetti
15 connessione.Terminator='CR/LF';
16 connessione.InputBufferSize=512;
17 connessione.OutputBufferSize=512;
18 %apro la connessione
19 fopen(connessione);
20 %pausa per dare tempo al robot di rispondermi positivamente
21 pause(0.01)
22 %leggo la risposta sull'ingresso del nodo Listen, qui compare l'intero pacc
23 fscanf(connessione,'%s')
24 end
```

La funzione non restituisce l'oggetto connessione, ma non necessita di dati in ingresso. Per un uso ottimale andrebbe scritta da sola in uno script separato, così da poter usare altri script diversi senza modificare la connessione.

Nella prima riga, tramite il comando tcpip, si crea l'oggetto connessione. Successivamente si definiscono i parametri come descritto al capitolo precedente ed infine si scannerizza la risposta personalizzata nel nodo Listen del diagramma di flusso.

3.5 Funzione Ask

La funzione Ask si occupa di chiedere le informazioni accessibili al robot, ovvero le coordinate nei giunti e nello spazio operativo, velocità del Tool e forza applicata

dal Tool.

Listing 3.5: Funzione Ask.m per ottenere informazioni dal robot

```
function q = ask(connessione,id)
%{
Vado a Chiedere informazioni al TM sul suo stato
Connessione Indica La variabile connessione che sto usando per connettermi
al TM attraverso TCP-ip

info indica invece cosa vado a chiedere: in questo informazioni sul
robot[0], per più info leggere il capitolo 6 del manuale Expression Editor
and Listen Node

1=Coordinate Joints da 1 a 6 in gradi[J1,J2,J3,J4,J5,J6]
2=Coordinate TCP in mm e gradi [x,y,z,rx,ry,rz]
3= Forza del TCP rispetto alla base
4= Velocità assoluta del TCP in mm/s

%}


%% caso 1: Coordinate Joints
switch id
    case 1

comando='1,ListenSend(90,Robot[0].Joint)';% pacchetto da inviare al TM
pack=writepack('TMSCT',comando);%funzione che scrive il pacchetto totale
fprintf(connessione,pack);%invio il pacchetto

d=fscanf(connessione);% Pacchetto di ritorno contenente le coordinate
k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
msg=d((end-29):(end-6)); %tolgo ,*CS dalla fine e conto 24 caratteri,
% 4 caratteri per valore sono le coordinate in single ( 8bit*4=32 bit))


A=double(msg); % trasformo in double i vari caratteri

%estraggo i valori dei Ji: ogni 4 caratteri (4*8=32bit) sono il Valore in
%Single
J1=single((typecast(uint8(A(1,1:4)), 'single')));
J2=single((typecast(uint8(A(1,5:8)), 'single')));
J3=single((typecast(uint8(A(1,9:12)), 'single')));
J4=single((typecast(uint8(A(1,13:16)), 'single')));
J5=single((typecast(uint8(A(1,17:20)), 'single')));
J6=single((typecast(uint8(A(1,21:24)), 'single')));



```

```
46 q=[ J1,J2,J3, J4, J5, J6];%unisco i Ji nel vettore q
47
48
49
50 %% caso 2: Coordinate Robot
51
52    case  2
53
54 comando='1,ListenSend(90,Robot[0].CoordRobot)';% pacchetto da inviare al TM
55 pack=writepack('TMSCT',comando);%funzione che scrive il pacchetto totale
56 fprintf(connessione,pack);%invio il pacchetto
57
58 d=fscanf(connessione);% Pacchetto contenente le coordinate
59 k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
60 msg=d((end-29):(end-6)); %tolgo ,*CS dalla fine e conto 24 caratteri,
61 % 4 caratteri per valore sono le coordinate in single ( 8bit*4=32 bit))
62
63
64 A=double(msg); % trasformo in double i vari caratteri
65
66 %estraggo i valori delle coordinate: ogni 4 caratteri (4*8=32bit) sono il
67 %Valore in Single
68 x=single((typecast(uint8(A(1,1:4)), 'single')));
69 y=single((typecast(uint8(A(1,5:8)), 'single')));
70 z=single((typecast(uint8(A(1,9:12)), 'single')));
71 rx=single((typecast(uint8(A(1,13:16)), 'single')));
72 ry=single((typecast(uint8(A(1,17:20)), 'single')));
73 rz=single((typecast(uint8(A(1,21:24)), 'single')));
74
75 q=[ x,y,z, rx, ry, rz];% unisco le singole coordinate
76
77
78 %% caso 3: TCPForce3D
79     case 3
80
81 comando='1,ListenSend(90,Robot[0].TCPForce3D)';% pacchetto da inviare al TM
82 pack=writepack('TMSCT',comando);%funzione che scrive il pacchetto totale
83 fprintf(connessione,pack);%invio il pacchetto
84
85 d=fscanf(connessione);% Pacchetto contenente le coordinate
86 k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
87 msg=d((end-9):(end-6)); %tolgo ,*CS dalla fine e conto 4 caratteri,
88 % 4 caratteri per valore sono le coordinate in single ( 8bit*4=32 bit))
89
90
91 A=double(msg); % trasformo in double i vari caratteri
92
93
```

```
94  F=single((typecast(uint8(A(1,1:4)), 'single')));%da double a Single
95
96
97  q= F;
98
99
100
101 %% caso 4: TCPSpeed3D
102     case 4
103
104 comando='1,ListenSend(90,Robot[0].TCPSpeed3D)';% pacchetto da inviare al TM
105 pack=writepack('TMSCT',comando);%funzione che scrive il pacchetto totale
106 fprintf(connessione,pack);%invio il pacchetto
107
108 d=fscanf(connessione);% Pacchetto contenente le coordinate
109 k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
110 msg=d((end-9):(end-6)); %tolgo ,*CS dalla fine e conto 4 caratteri,
111 % 4 caratteri per valore sono le coordinate in single ( 8bit*4=32 bit))
112
113
114 A=double(msg); % trasformo in double i vari caratteri
115
116
117 S=single((typecast(uint8(A(1,1:4)), 'single')));%da double a Single
118
119
120 q=S;
121
122
123 end
```

Le variabili di ingresso sono: connessione, ovvero l'oggetto connessione che unisce robot e pc ed un id che definisce i 4 dati che si possono ottenere con la funzione Ask.

- 1: Coordinate joints
- 2: Coordinate TCP in mm e gradi
- 3: Forza esercitata dal TCP rispetto alla base
- 4: Velocità assoluta del TCP in mm/s

3.5.1 Caso 1: Coordinate Joints

Per richiedere informazioni si utilizza il comando ListenSend: così si chiede al nodo listen di restituire l'informazione richiesta al suo interno, ad esempio nel caso dei joints si scrive:

$1, ListenSend(90, Robot[0].Joint)$ dove:

- 1 è l'id del comando
- ListenSend La funzione
- 90 è un id del ListenSend: è necessario affinché funzioni tutto correttamente
- Robot[0].Joints è l'argomento richiesto.

si scrive dunque il pacchetto tramite la funzione writepack definita precedentemente, tramite $fprintf(connessione, pack)$ la si invia al robot.

A questo punto si riceve la risposta: con un primo comando $fscanf(connessione)$ si memorizza il pacchetto in ingresso contenente le coordinate, con un secondo $fscanf(connessione)$ si legge il messaggio di ok ricevuto, senza che questo sia visualizzato: è importante non saltare questo passaggio poiché l'ok rimarrebbe in coda andando a disturbare la corretta esecuzione di altri comandi.

Il pacchetto con le coordinate joints viene visualizzato come un insieme di caratteri non riconosciuti e caratteri ascii:

```
%EsempioPacchetto '$TMSTA,27,90,Ð[ÝC. tC?ÏìC·?3CØ,□A□t□C,*54'
```

Figura 3.1: Esempio Joints

Questo perché la parte di pacchetto accessoria è data in codice ascii, mentre le sei coordinate di giunto sono espresse come single, associando 32 bit, ovvero 4 byte ovvero 4 caratteri, ad ogni valore.

il passo successivo è quello di prendere solo i caratteri che corrispondono alle coordinate e salvarli nella variabile msg questa poi viene trasformata in double.

Attraverso la funzione:

$$J1 = single((typecast(uint8(A(1, 1:4)), 'single')));$$

si isolano i primi 4 byte e vengono trasformati in un valore single, memorizzato in J1. Si ripete per ogni valore di J_i e si uniscono poi nel vettore q chela funzione ask restituisce.

3.5.2 Caso 2: Coordinate TCP

Nel caso 2, si vogliono sapere le coordinate del TCP. Il procedimento è completamente uguale al caso 1, con l'unica differenza nel comando che viene modificato in 1, $ListenSend(90, Robot[0].CoordRobot)$

Anche in questo caso si incontra lo stesso problema di visualizzazione delle coordinate e tramite le stesse esatte formule si ottengono le coordinate $[X, Y, Z, rx, ry, rz]$ espresse secondo la notazione di Cardano.

3.5.3 Caso 3: Forza TCP

Nel caso 3 cerco di ottenere il valore di forza TCP.Questo è un valore unico e solo relativo alla base del robot, il comando è 1, $ListenSend(90, Robot[0].TCPForce3D)$

Il processo è analogo ai precedenti, solamente che i dati sono racchiusi in 4 byte anziché 24.

3.5.4 Caso 4: Velocità TCP

Nel caso 4 richiedo il valore di velocità del TCP in mm/s.Questo è un valore unico, il comando è 1, $ListenSend(90, Robot[0].TCPSpeed3D)$

Il processo è analogo al precedente.

3.6 ReadIO

La funzione ReadIO serve per leggere i valori degli output digitali e analogici.

Listing 3.6: Funzione ReadIO.m per ottenere il valore di input digitali e analogici

```
1 function I=ReadIO(connessione,fonte,tipo,id)
2 %la funzione di ReadIO restituisce  il valore richiesto con
3 %il comando che viene messo in lista d'attesa all'interno del robot.
4
5 %esempio
```

```
%comando='1,IO["ControlBox"].DO[1]';

%connessione è il nome della connessione
%fonte degli output, di base posso scegliere tra "ControlBox" oppure
%"EndModule"
%tipo è DI / AI, se provo a leggere un input dovrebbe darmi errore
% id è il numero di input, per i digital 0–15 per gli analog da 0 a
%


% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,ListenSend(90,IO["',string(fonte),'"].',string(tipo),'[',string(id),'])'));
pack=writepack('TMSCT',comando);
fprintf(connessione,pack);

d=fscanf(connessione);% Pacchetto contenente il valore di output
k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
msg=d(end–6); %tolgo ,*CS dalla fine e conto 4 caratteri,
% 4 caratteri per valore sono le info in single ( 8bit*4=32 bit))

A=double(msg); % trasformo in double i vari caratteri


I=single(uint8(A(1)));









end
```

Le variabili di ingresso sono:

- 1: Connessione, che corrisponde con l'oggetto tcpip
- 2: Fonte, che può essere "ControlBox" oppure "EndModule" nel caso della stazione in laboratorio, se invece sono presenti anche moduli di espansione si inserisce il nome del modulo.
- 3: Tipo, per distinguere Digital da Analog
- 4: id, che va da 0 a 15 per il ControlBox e 0-1-2 per EndModule

Come prima operazione si compone il comando partendo dai dati in ingresso e successivamente si compone e si invia il pacchetto.

Successivamente tramite $fscanf$ si rileva la riposta con i dati richiesti e con un successivo $fscanf$ si ascolta la riposta di ok, altrimenti rimarrebbe in coda andando a disturbare il funzionamento di successivi comandi.

Per quando riguarda i digital input questi sono definiti da un singolo carattere, che tradotto in single diventa 0 oppure 1.

Per gli analogici non c'è stato modo di poterli testare dunque potrebbero non funzionare, nel caso si consiglia di controllare la struttura del pacchetto tramite il software wireshark in modo da vedere bene la struttura e posizione del dato del segnale analogico.

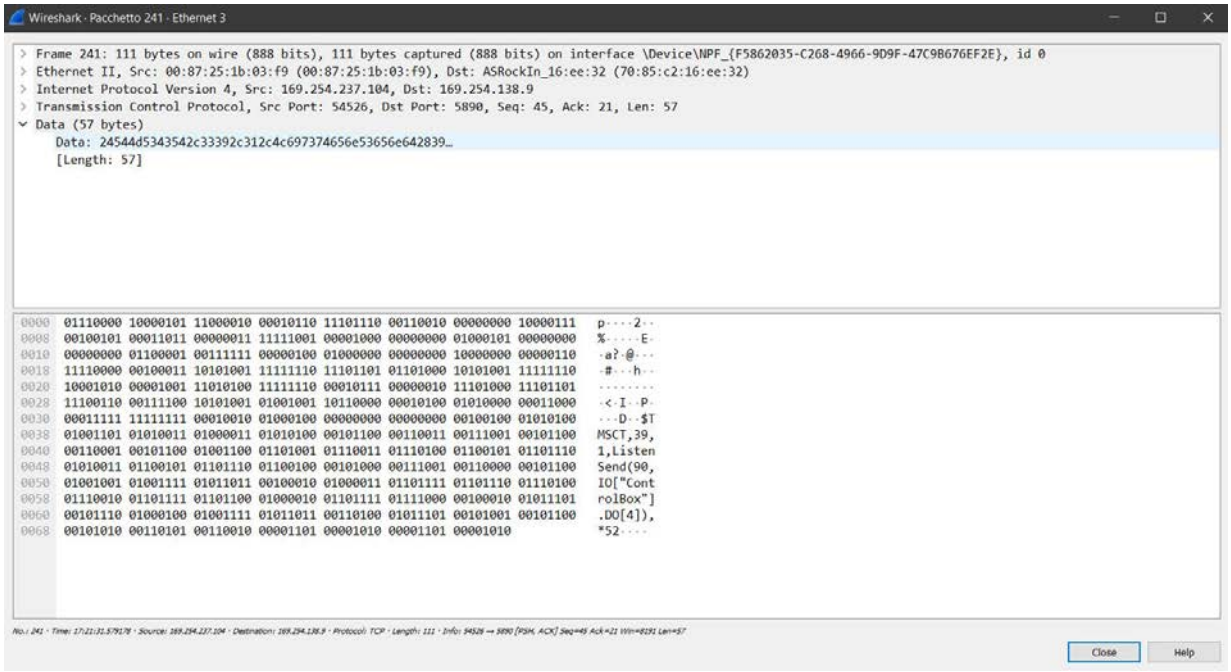


Figura 3.2: Pacchetto ReadIO inviato

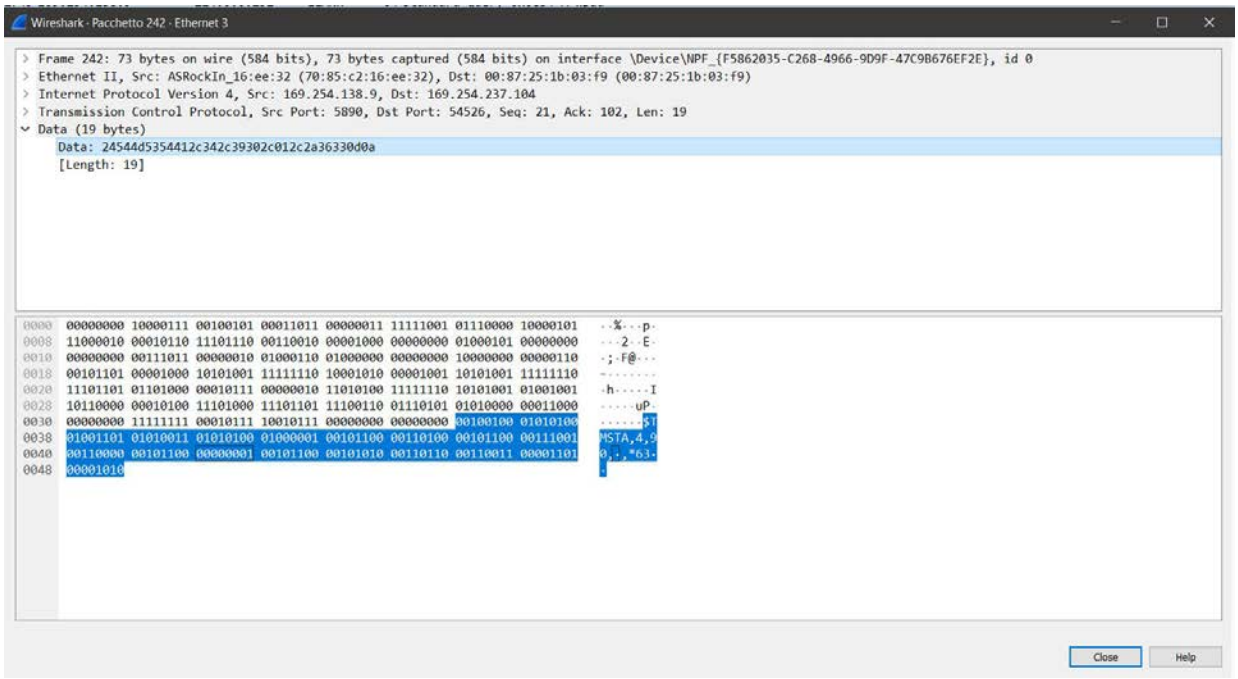


Figura 3.3: Pacchetto ReadIO ricevuto

3.7 WriteIO

La funzione WriteIO serve per scrivere i valori degli output digitali e analogici.

Listing 3.7: Funzione ReadIO.m per scrivere il valore di output digitali e analogici

```
function I=ReadIO(connessione,fonte,tipo,id)
%la funzione di ReadIO restituisce  il valore richiesto con
%il comando che viene messo in lista d'attesa all'interno del robot.

%esempio
%comando='1,IO["ControlBox"].DO[1]';

%connessione è il nome della connessione
%fonte degli output, di base posso scegliere tra "ControlBox" oppure
%"EndModule"
%tipo è DI / AI, se provo a leggere un input dovrebbe darmi errore
% id è il numero di input, per i digital 0–15 per gli analog da 0 a
%



% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,ListenSend(90,IO["',string(fonte),'"].',string(tipo),'[',string(id),'])'));
pack=writepack('TMSCT',comando);
fprintf(connessione,pack);
```

```
21
22 d=fscanf(connessione);% Pacchetto contenente il valore di output
23 k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
24 msg=d(end-6); %tolgo ,*CS dalla fine e conto 4 caratteri,
25 % 4 caratteri per valore sono le info in single ( 8bit*4=32 bit))
26
27 A=double(msg); % trasformo in double i vari caratteri
28
29
30 I=single(uint8(A(1)));
31
32
33
34
35
36
37 end
```

Le variabili di ingresso sono:

- 1: Connessione, che corrisponde con l'oggetto tcpip
- 2: Fonte, che può essere "ControlBox" oppure "EndModule" nel caso della stazione in laboratorio, se invece sono presenti anche moduli di espansione si inserisce il nome del modulo.
- 3: Tipo, per distinguere Digital da Analog
- 4: id, che va da 0 a 15 per il ControlBox e 0-1-2 per EndModule

Come prima operazione si compone il comando partendo dai dati in ingresso e successivamente si compone e si invia il pacchetto.

Successivamente tramite $fscanf$ si rileva la riposta con i dati richiesti e con un successivo $fscanf$ si ascolta la riposta di ok, altrimenti rimarrebbe in coda andando a disturbare il funzionamento di successivi comandi.

Per quando riguarda i digital output questi sono definiti da un singolo carattere, che tradotto in single diventa 0 oppure 1.

Per gli analogici non c'è stato modo di poterli testare dunque potrebbero non funzionare, nel caso si consiglia di controllare la struttura del pacchetto tramite il software wireshark in modo da vedere bene la struttura e posizione del dato del segnale analogico.

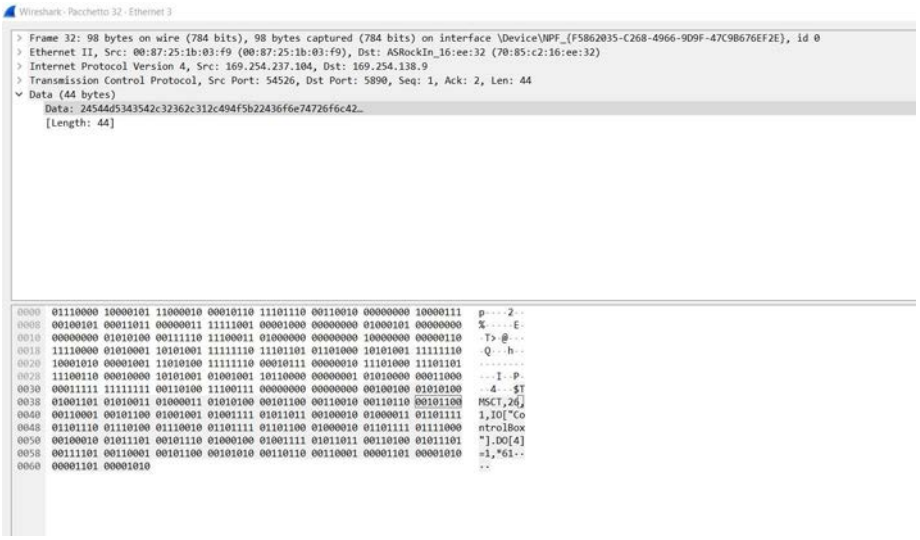


Figura 3.4: Pacchetto WriteIO

3.8 InstantReadIO

La funzione InstantReadIO serve per ottenere i valori degli output digitali e analogici. A differenza della funzione ReadIO questa non viene messa in coda agli altri comandi ma viene eseguita subito nell'istante in cui viene ricevuta.

Listing 3.8: Funzione ReadIO.m per ottenere istantaneamente il valore di output digitali e analogici

```
function I=InstantReadIO(connessione,fonte,tipo, id)
%% Descrizione
%la funzione di IstantReadIO restituisce SUBITO il valore richiesto senza
%che il comando sia messo in lista d'attesa all'interno del robot.

%esempio di comando che devo comporre
%comando='1,IO["ControlBox"].DO[1]';

%% INPUT
%connessione è il nome della connessione
%fonte degli output, di base posso scegliere tra "ControlBox" oppure
%"EndModule"
%tipo è DI / AI, se provo a leggere un input dovrebbe darmi errore
% id è il numero di input, per i digital 0—15 per gli analog da 0 a

% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,ListenSend(90,IO["',string(fonte),'"].Instant',string(tipo),'[',string(id),'])
    '));
```

```
19 pack=writepack('TMSCT',comando);
20 fprintf(connessione,pack);
21
22 d=fscanf(connessione);% Pacchetto contenente il valore di output
23 k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
24 msg=d(end-6); %tolgo ,*CS dalla fine e conto 4 caratteri,
25 % 4 caratteri per valore sono le info in single ( 8bit*4=32 bit))
26
27 A=double(msg); % trasformo in double i vari caratteri
28
29
30 I=single(uint8(A(1)));
```

Le variabili di ingresso sono:

- 1: Connessione, che corrisponde con l'oggetto tcpip
- 2: Fonte, che può essere "ControlBox" oppure "EndModule" nel caso della stazione in laboratorio, se invece sono presenti anche moduli di espansione si inserisce il nome del modulo.
- 3: Tipo, per distinguere Digital da Analog
- 4: id, che va da 0 a 15 per il ControlBox e 0-1-2 per EndModule

Come prima operazione si compone il comando partendo dai dati in ingresso e successivamente si compone e si invia il pacchetto.

Successivamente tramite $fscanf$ si rileva la riposta con i dati richiesti e con un successivo $fscanf$ si ascolta la riposta di ok, altrimenti rimarrebbe in coda andando a disturbare il funzionamento di successivi comandi.

Per quando riguarda i digital input questi sono definiti da un singolo carattere, che tradotto in single diventa 0 oppure 1.

Per gli analogici non c'è stato modo di poterli testare dunque potrebbero non funzionare, nel caso si consiglia di controllare la struttura del pacchetto tramite il software wireshark in modo da vedere bene la struttura e posizione del dato del segnale analogico.

3.9 IstantWriteIO

La funzione IstantWriteIO serve per scrivere istantaneamente i valori degli output digitali e analogici.

A differenza della funzione WriteIO questa non viene messa in coda agli altri comandi ma viene eseguita subito nell'istante in cui viene ricevuta.

Listing 3.9: Funzione ReadIO.m per scrivere il valore di output digitali e analogici

```
function I=ReadIO(connessione,fonte,tipo,id)
%la funzione di ReadIO restituisce  il valore richiesto con
%il comando che viene messo in lista d'attesa all'interno del robot.

%esempio
%comando='1,IO["ControlBox"].DO[1]';

%connessione è il nome della connessione
%fonte degli output, di base posso scegliere tra "ControlBox" oppure
%"EndModule"
%tipo è DI / AI, se provo a leggere un input dovrebbe darmi errore
% id è il numero di input, per i digital 0—15 per gli analog da 0 a
%


% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,ListenSend(90,IO["',string(fonte),'"].',string(tipo),'[',string(id),'])'));
pack=writepack('TMSCT',comando);
fprintf(connessione,pack);

d=fscanf(connessione);% Pacchetto contenente il valore di output
k=fscanf(connessione);% Messaggio di OK, non viene utilizzato come dato
msg=d(end—6); %tolgo ,*CS dalla fine e conto 4 caratteri,
% 4 caratteri per valore sono le info in single ( 8bit*4=32 bit))

A=double(msg); % trasformo in double i vari caratteri


I=single(uint8(A(1)));









end
```

Le variabili di ingresso sono:

- 1: Connessione, che corrisponde con l'oggetto tcpip
- 2: Fonte, che può essere "ControlBox" oppure "EndModule" nel caso della stazione in laboratorio, se invece sono presenti anche moduli di espansione si inserisce il nome del modulo.
- 3: Tipo, per distinguere Digital da Analog
- 4: id, che va da 0 a 15 per il ControlBox e 0-1-2 per EndModule

Come prima operazione si compone il comando partendo dai dati in ingresso e successivamente si compone e si invia il pacchetto.

Successivamente tramite $fscanf$ si rileva la riposta con i dati richiesti e con un successivo $fscanf$ si ascolta la riposta di ok, altrimenti rimarrebbe in coda andando a disturbare il funzionamento di successivi comandi.

Per quando riguarda i digital Output questi sono definiti da un singolo carattere, che tradotto in single diventa 0 oppure 1.

Per gli analogici non c'è stato modo di poterli testare dunque potrebbero non funzionare, nel caso si consiglia di controllare la struttura del pacchetto tramite il software wireshark in modo da vedere bene la struttura e posizione del dato del segnale analogico.

3.10 Funzione PTP

Passiamo dunque alle funzioni di movimento. La prima che viene descritta è la funzione per un movimento point to point, ovvero PTP.

Listing 3.10: Funzione PTP.m per eseguire un movimento point to point

```
1 function answer=PTP(connessione,txt,coord,vel,Ta,racc,PrecPositioning,conf )
2 %% PTP point 2 point
3 %{
4
5
6 Effettuo il movimento Point 2 Point dal punto attuale a quello di arrivo.
7 per info guardare il manuale Expression_Editor_and_Listen_Node del TM.
8
```

```
id,PTP(0,1,2,3,4,5,6,7)

0:Nome della connessione

1: una string che definisce il formato: 3 lettere
       1: Motion target format:

        J expressed in joint angles
        C expressed in Cartesian coordinate
       2: Speed format:
        P expressed as a percentage
       3: Blending format
        P expressed as a percentage

2: coordinate, di giunto in ordine J 1—2—3—4—5—6 in gradi
   oppure cartesiane assolute: x,y,x in mm e Rx Ry Rz
3: velocità in %
le velocità scalano con quella che è la % di Progetto Listen
4: tempo per accelerare a vmax ( Ta) in ms
5: unione traiettorie ( in % )
6: Disable precise positioning ( boolean )
            1:true
                Disable precise positioning
            2:false
                Enable precise positioning

7: se uso coordinate Cartesiane specifico la config del robot
   in un vettore {0—1,2—3,4—5}


        0:Braccio destro
        1 Braccio Sinistro

        2: Gomito Alto
        3: Gomito Basso

        4: polso NON flippato
        5: polso Flippato
%}
%esempio

%esprimo le coordinate contenute in 'coord' come string e le salvo nella
%stringa 'coo'
coo=strcat('{',string(coord(1)),',',string(coord(2)),',',string(coord(3)),',',string(coord(4)),',',
     string(coord(5)),',',string(coord(6)),'}');

%se nargin>7 ho anche la configurazione e devo adattarla dal file config a
% un vettore come descritto sopra.
```

```
56  if nargin >7
57
58  if conf.righty == 1
59      a=0;
60  else
61      a=1;
62  end
63
64  if conf.below == 1
65      b=3;
66  else
67      b=2;
68  end
69
70  if conf.flip == 1
71      c=5;
72  else
73      c=4;
74  end
75
76  % config diventa un file string espresso come '{0-1,2-3,4-5}'
77  config=strcat('{',string(a),',',string(b),',',string(c),'}');
78
79  % scrivo il comando, deve risultare in char. La funzione strcat() richiede
80  % String o Char, per questo trasformo tutto in String
81  comando=char(strcat('1,PTP("',txt,'",',char(coo),',',string(vel),',',string(Ta),',',string(racc),',',
        string(PrecPositioning),',',config,')'));
82
83  % esempio comando
84  %comando='1,PTP("CPP",{400,300,300,180,0,180},10,500,50,false,{1,2,4})';
85
86
87
88  else  % caso con nargin =7
89
90
91
92
93  comando=char(strcat('1,PTP("',txt,'",',char(coo),',',string(vel),',',string(Ta),',',string(racc),',',
        string(PrecPositioning),')'));
94
95  %esempi comando
96  %comando='1,PTP("JPP",{61.7399,286.7598,-106.7598,90.0000,-90.0000,-28.2601},15,500,0,false )';
97  %comando='1,PTP("JPP",0,0,0,0,0,0,25,500,10,false)'% anche va bene
98  %comando='1,PTP("CPP",500,500,600,180,-1,180,10,500,1,true,1,2,4)'
99  end
100
101 pack=writepack('TMSCT',comando);
```

```
102 fprintf(connessione,'%s',pack);
103 answer=fscanf(connessione)
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot
- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:
  1. Formato Coordinate destinazione: C: per Cartesiano J: per Joints
  2. Formato velocità: P: Per Percentuale
  3. Formato Raccordo tra i movimenti P: Per Percentuale
- coordinate, espresse come vettore in riga $[J_1, J_2, J_3, J_4, J_5, J_6]$ oppure $[X, Y, Z, rx, ry, rz]$
- Velocità in %, le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.
- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- Raccordo tra i movimenti in %
- Valore booleano,se disattivare o meno il precise positioning
- Configurazione, valido solo se in coordinate cartesiane:
  - $conf.righty = 1$ se braccio destro,$conf.righty = 0$ se braccio sinistro
  - $conf.below = 1$ se gomito basso,$conf.below = 0$ se gomito alto
  - $conf.flip = 1$ se polso flippato,$conf.flip = 0$ se polso non flippato

Come primo passo si trasforma il vettore di coordinate in ingresso in un vettore che formalmente è uguale, ma il tipo di variabile risulta essere string, in modo che si possa concatenare con il resto del comando.

Si distingue subito il caso in cui ci si muove nello spazio cartesiano o in quello dei giunti perché il numero di variabili è diverso. Nel primo caso si traduce la variabile $conf$ in un vettore$\{0-1, 2-3.4-5\}$ per poterlo inserire nella stringa

di comando. Si forma quindi il la stringa con il comando. Se si opera nello spazio dei giunti si procede subito con la compilazione del comando.

In entrambi i casi si va a comporre dal comando il pacchetto tramite la funzione writepack, lo si invia e si riceve la risposta di conferma. il robot a questo punto dovrebbe muoversi.

3.11 Funzione Line

La funzione Line permette di compiere un movimento lungo una linea retta.

Listing 3.11: Funzione Line.m per eseguire un movimento lineare

```
function answer=Line(connessione,txt,coord,speed,Ta,racc,PrecPos)
%% movimento line
%{

Effettuo il movimento Line dal punto attuale a quello di arrivo.
per info guardare il manuale Expression_Editor_and_Listen_Node del TM.


id,Line(0,1,2,3,4,5,6)
0:nome connessione
1: una string che definisce il formato: 3 lettere
       1: Motion target format:
        C expressed in Cartesian coordinate
       2: Speed format:
        P expressed as a percentage
        A expressed in velocity (mm/s)
       le velocità scalano con quella che è la % di Progetto Listen
       3: Blending format
        P expressed as a percentage
        R expressed as radius
2: coordinate cartesiane assolute: x,y,x in mm e Rx Ry Rz
3: velocità in % o in mm/s
4: tempo per accelerare a vmax ( Ta) in ms
5: unione traiettorie ( in % ) o in raggio (mm)
6: Disable precise positioning ( boolean )
            1:true
                Disable precise positioning
            2:false
                Enable precise positioning


%}

```

```
%esprimo le coordinate contenute in 'coord' come string e le salvo nella
%stringa 'coo'
coo=strcat('{',string(coord(1)),',',string(coord(2)),',',string(coord(3)),',',string(coord(4)),',',
     string(coord(5)),',',string(coord(6)),'}');

% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,Line("',txt,'",',char(coo),',',string(speed),',',string(Ta),',',string(racc),'
     ,',string(PrecPos),')'));

%esempio comando
%comando='1,Line("CAR",{400,400,400,180,0,180},10,100,10,false)';

pack=writepack('TMSCT',comando);

fprintf(connessione,'%s',pack);
answer=fscanf(connessione);
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot
- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:
  1. Formato Coordinate destinazione: C: per Cartesiano
  2. Formato velocità: P: Per Percentuale A: per assoluta in mm/s
  3. Formato Raccordo tra i movimenti P: Per Percentuale R: radius
- coordinate, espresse come vettore in riga$[X, Y, Z, rx, ry, rz]$
- Velocità in % o in mm/s,le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.
- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- Raccordo tra i movimenti, in % oppure in mm
- Valore booleano,se disattivare o meno il precise positioning

Come primo passo si trasforma il vettore di coordinate in ingresso in un vettore che formalmente è uguale, ma il tipo di variabile risulta essere string, in modo che

si possa concatenare con il resto del comando. Si compone dunque il comando il pacchetto tramite la funzione writepack, lo si invia e si riceve la risposta di conferma. il robot a questo punto dovrebbe muoversi.

3.12 Funzione Pline

La funzione PLine permette di compiere un movimento Pline, il raccordo funziona in modalità diversa rispetto al line normale. Cercando la funzione Pline sul software TMFlow non c'è un nodo specificato, perché PLine è una opzione di movimento del nodo Path, che viene usato in caso di percorso prefissato formato da vari punti.

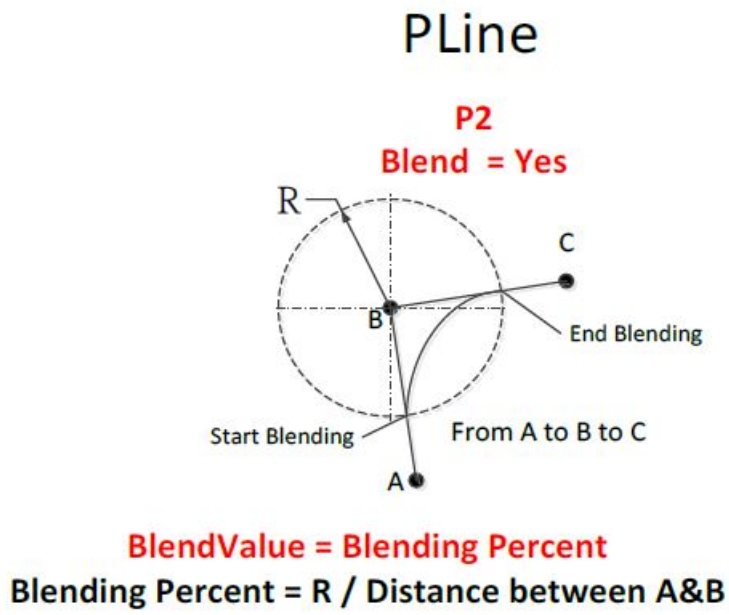


Figura 3.5: Raccordo PLine

Listing 3.12: Funzione PLine.m per eseguire un movimento Pline

```
1  function  answer=PLine(connessione,txt,coord,vel,Ta,racc)
2
3  %% movimento Pline
4  %{
5
6  Effettuo il movimento PLine dal punto attuale a quello di arrivo.
7  per info guardare il manuale Expression_Editor_and_Listen_Node del TM.
```

```
8
9   id,PLine(0,1,2,3,4,5)
10  0:nome della connessione
11  1: una string che definisce il formato: 3 lettere
12         1: Motion target format:
13          J: expressed in joint angles
14          C expressed in Cartesian coordinate
15         2: Speed format:
16          A: expressed in velocity (mm/s)
17
18
19         3: Blending format
20          P expressed as a percentage
21
22  2: coordinate cartesiane assolute: x,y,x in mm e Rx Ry Rz
23     oppure in gradi di J 1–2–3–4–5–6
24  3: velocità in mm/s
25  4: tempo per accelerare a vmax ( Ta) in ms
26  5: unione traiettorie ( in % )
27
28
29
30  %}
31  %esempio
32
33  %esprimo le coordinate contenute in 'coord' come string e le salvo nella
34  %stringa 'coo'
35  coo=strcat('{',string(coord(1)),',',string(coord(2)),',',string(coord(3)),',',string(coord(4)),',',
        string(coord(5)),',',string(coord(6)),'}');
36
37  % scrivo il comando, deve risultare in char. La funzione strcat() richiede
38  % String o Char, per questo trasformo tutto in String
39  comando=char(strcat('1,PLine("',txt,'",',char(coo),',',string(vel),',',string(Ta),',',string(racc),')
        '));
40
41  %esempi
42  %comando='1,PLine("JAP",{0,0,0,0,0,0},5,500,0)';
43  %comando='1,PLine("CAP",{500,350,500,180,0,180},25,500,0)';
44
45  pack=writepack('TMSCT',comando);
46
47  fprintf(connessione,'%s',pack);
48  answer=fscanf(connessione);
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot

- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:

  1. Formato Coordiante destinazione: C: per Cartesiano J: per Joints
  2. Formato velocità:

     A: per assoluta in mm/s
  3. Formato Raccordo tra i movimenti P: Per Percentuale

- coordinate, espresse come vettore in riga $[J_1, J_2, J_3, J_4, J_5, J_6]$ oppure $[X, Y, Z, rx, ry, rz]$
- Velocità in % o in mm/s,le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.
- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- Raccordo tra i movimenti, in % oppure in mm

Come primo passo si trasforma il vettore di coordinate in ingresso in un vettore che formalmente è uguale, ma il tipo di variabile risulta essere string, in modo che si possa concatenare con il resto del comando. Si compone dunque il comando il pacchetto tramite la funzione writepack, lo si invia e si riceve la risposta di conferma. il robot a questo punto dovrebbe muoversi.

3.13 Funzione Circle

La funzione circle viene usata quando si vuole percorrere una traiettoria che sia esattamente circolare.

Listing 3.13: Funzione Circle.m per eseguire un movimento a forma circolare

```
function answer=Circle(connessione,txt,puntomezzo,puntofin,speed,Ta,racc,ArcAngle,PrecPos)

%%  Movimento CIRCLE

%{

id,Cricle(1,2,3,4,5,6,7,8)
0:nome connesssione
1: una string che definisce il formato: 3 lettere
```

```
      1: Motion target format:
       C expressed in Cartesian coordinate
      2: Speed format:
       P expressed as a percentage
  le velocità scalano con quella che è la % di Progetto Listen
       A expressed in velocity (mm/s)
      3: Blending format
       P expressed as a percentage
2: coordinate cartesiane assolute di un punto dell'arco: x,y,x in mm e Rx Ry Rz
3: coordinate cartesiane assolute del punto finale: x,y,x in mm e Rx Ry Rz
4: velocità in % o in mm/s
5: tempo per accelerare a vmax ( Ta) in ms
6: unione traiettorie ( in % )
7:Arc angle(°°), If non-zero value is given, the TCP will keep the same pose and move from
  current point to the assigned arc angle via the given point and end point on arc;
  If zero is given, the TCP will move from current point and pose to end point
  and pose via the point on arc with linear interpolation on pose.
8:Disable precise positioning ( boolean )
           1:true
               Disable precise positioning
           2:false
               Enable precise positioning

%}
% i punti devono essere espressi nella forma {1,2,3,4,5,6}, in char, in
% modo che si possano contare le lettere
pm=strcat('{',string(puntomezzo(1)),',',string(puntomezzo(2)),',',string(puntomezzo(3)),',',string(
    puntomezzo(4)),',',string(puntomezzo(5)),',',string(puntomezzo(6)),'}');
pf=strcat('{',string(puntofin(1)),',',string(puntofin(2)),',',string(puntofin(3)),',',string(puntofin
    (4)),',',string(puntofin(5)),',',string(puntofin(6)),'}');

% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,Circle("',txt,'",',char(pm),',',char(pf),',',string(speed),',',string(Ta),',',
    string(racc),',',string(ArcAngle),',',string(PrecPos),')'));

%esempio di comando
%comando='1,Circle("CAP",{300,300,400,180,0,180},{200,400,400,180,0,180},25,500,10,0,false)';

pack=writepack('TMSCT',comando);%scrivo pacchetto

fprintf(connessione,'%s',pack);%invio pacchetto
answer=fscanf(connessione)%risposta di ok
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot

- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:

  1. Formato Coordinate destinazione: C: per Cartesiano
  2. Formato velocità: P: Per Percentuale A: per assoluta in mm/s
  3. Raccordo tra i movimenti P: Per Percentuale

- coordinate cartesiane di un punto interno dell'arco espresse come $[X, Y, Z, rx, ry, rz]$
- coordinate cartesiane del punto finale dell'arco espresse come $[X, Y, Z, rx, ry, rz]$
- Velocità in % o in mm/s,le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.
- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- Raccordo tra i movimenti, in %
- Arc angle: se un valore diverso da zero, il tcp mantiene la stessa posa e si muove dalla posizione corrente dell'angolo assegnato lungo l'arco; se invece è zero il tcp si muove dalla posizione attuale a quella finale.

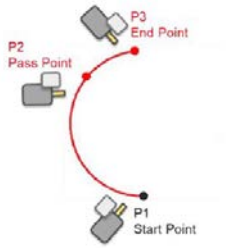


Figura 3.6: Circle con angle arc = 0

- valore booleano per la disattivazione del precise positioning

Il punto in mezzo e finale vengono tradotti in una stringa e da questi si compone poi il comando. Tramite la funzione writepack si scrive il pacchetto finale,lo si invia e si riceve la risposta di ok tramite fscanf. Il robot dovrebbe cosi muoversi come richiesto.

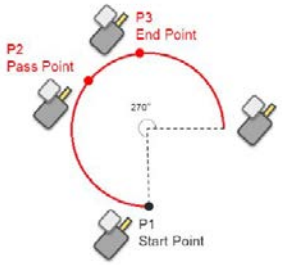


Figura 3.7: Circle con arc angle =270

3.14 Funzione Move_PTP

Passiamo dunque alle funzioni di movimento relativo alla posizione attuale. La prima che viene descritta è la funzione per un movimento point to point, ovvero Move_PTP.

Listing 3.14: Funzione Move_PTP.m per eseguire un movimento relativo point to point

```
function answer = Move_PTP(connessione,txt, coord,vel,Ta,racc,PrecPos,conf)
%% move_PTP point 2 point
%Effettuo il movimento RELATIVO Point 2 Point dal punto attuale a quello di arrivo.
%per info guardare il manuale Expression_Editor_and_Listen_Node del TM.

%{
id,PTP(1,2,3,4,5,6)
0:nome connessione
1: una string che definisce il formato: 3 lettere
       1: Motion target format:
        J expressed in joint angles
        C expressed in Cartesian coordinate
       2: Speed format:
        P expressed as a percentage
       3: Blending format
        P expressed as a percentage

2: coordinate, di giunto in ordine J 1_2_3_4_5_6 in gradi
   oppure cartesiane assolute: x,y,x in mm e Rx Ry Rz
3: velocità in %
le velocità scalano con quella che è la % di Progetto Listen
4: tempo per accelerare a vmax ( Ta) in ms
5: unione traiettorie ( in % )
6: Disable precise positioning ( boolean )
            1:true
                Disable precise positioning
```

```
            2:false
                Enable precise positioning

7: se uso coordinate Cartesiane specifico la config del robot
   in un vettore {0—1,2—3,4—5}


        0:Braccio destro
        1 Braccio Sinistro

        2: Gomito Alto
        3: Gomito Basso

        4: polso NON flippato
        5: polso Flippato
%}
%esempio

%solo
%esprimo le coordinate contenute in 'coord' come string e le salvo nella
%stringa 'coo'
coo=strcat('{',string(coord(1)),',',string(coord(2)),',',string(coord(3)),',',string(coord(4)),',',
     string(coord(5)),',',string(coord(6)),'}');

% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,Move_PTP("',txt,'",',char(coo),',',string(vel),',',string(Ta),',',string(racc)
     ,',',string(PrecPos),')'));

pack=writepack('TMSCT',comando);
fprintf(connessione,'%s',pack);
answer=fscanf(connessione);
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot
- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:
  1. Formato Coordinate Destinazione: C: per Cartesiano J: per Joints
  2. Formato velocità: P: Per Percentuale
  3. Formato Raccordo Traiettorie P: Per Percentuale
- coordinate, espresse come vettore in riga $[J_1, J_2, J_3, J_4, J_5, J_6]$ oppure $[X, Y, Z, rx, ry, rz]$

- Velocità in %, le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.
- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- Raccordo Traiettorie, in %
- Valore booleano,se disattivare o meno il precise positioning

Come primo passo si trasforma il vettore di coordinate in ingresso in un vettore che formalmente è uguale, ma il tipo di variabile risulta essere string, in modo che si possa concatenare con il resto del comando.

Si va a scrivere il pacchetto tramite la funzione writepack, lo si invia e si riceve la risposta di conferma. il robot a questo punto dovrebbe muoversi.

3.15 Funzione Move_Line

La funzione Move_Line permette di compiere un movimento relativo lungo una linea retta.

Listing 3.15: Funzione Move_Line.m per eseguire un movimento relativo lineare

```
function answer = Move_Line(connessione,txt,coord,vel,Ta,racc,PrecPos)
%% movimento relativo Line
%Effettuo il movimento RELATIVO Line dal punto attuale a quello di arrivo.
%per info guardare il manuale Expression_Editor_and_Listen_Node del TM.
%{

id,Move_Line(1,2,3,4,5,6)
0: nome connessione
1: una string che definisce il formato: 3 lettere
       1: Motion target format:
        C expressed in Cartesian coordinate
        T: expressed w.r.t. tool coordinate
       2: Speed format:
        P expressed as a percentage
        A expressed in velocity (mm/s)
       3: Blending format
        P expressed as a percentage
        R expressed as radius
2: coordinate cartesiane relative alla base: x,y,x in mm e Rx Ry Rz
   coordinate relative al tool
3: velocità in % o in mm/s
```

```
4: tempo per accelerare a vmax ( Ta) in ms
5: unione traiettorie ( in % ) o in raggio (mm)
6: Disable precise positioning ( boolean )
            1:true
                Disable precise positioning
            2:false
                Enable precise positioning


%}



%esprimo le coordinate contenute in 'coord' come string e le salvo nella
%stringa 'coo'
coo=strcat('{',string(coord(1)),',',string(coord(2)),',',string(coord(3)),',',string(coord(4)),',',
     string(coord(5)),',',string(coord(6)),'}');
% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,Move_Line("',txt,'",',char(coo),',',string(vel),',',string(Ta),',',string(racc
     ),',',string(PrecPos),')'));

%esempio
%comando='1,Move_Line("CPP",{-50,0,0,0,0,0},25,500,10,false)';
pack=writepack('TMSCT',comando);

fprintf(connessione,'%s',pack);
answer=fscanf(connessione);
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot
- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:
    1. Formato Coordinate destinazione: C: per Cartesiano T: per terna tool
    2. Formato velocità: P: Per Percentuale A: per assoluta in mm/s
    3. Formato unione traiettorie P: Per Percentuale R: radius
- coordinate, espresse come vettore in riga$[X, Y, Z, rx, ry, rz]$
- Velocità in % o in mm/s,le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.

- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- unione tra le traiettorie, in % oppure in mm
- Valore booleano,se disattivare o meno il precise positioning

Come primo passo si trasforma il vettore di coordinate in ingresso in un vettore che formalmente è uguale, ma il tipo di variabile risulta essere string, in modo che si possa concatenare con il resto del comando. Si compone dunque il comando il pacchetto tramite la funzione writepack, lo si invia e si riceve la risposta di conferma. il robot a questo punto dovrebbe muoversi.

3.16 Funzione Move_Pline

La funzione Move_PLine permette di compiere un movimento Pline, il raccordo funziona in modalità diversa rispetto al line normale. Cercando la funzione Move_PLine sul software TMFlow non c'è un nodo specificato, perché PLine è una opzione di movimento del nodo Path, che viene usato in caso di percorso prefissato formato da vari punti.

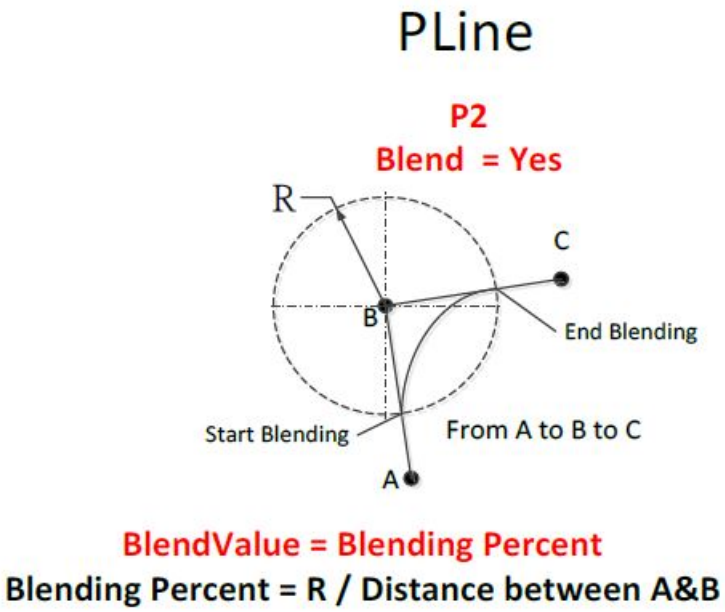


Figura 3.8: Raccordo Move_PLine

Listing 3.16: Funzione Move_Pline.m per eseguire un movimento relativo Pline

```
function  answer=Move_PLine(connessione,txt,coord,vel,Ta,racc)

%% movimento  Move_Pline
%{
%Effettuo il movimento RELATIVO PLine dal punto attuale a quello di arrivo.
%per info guardare il manuale Expression_Editor_and_Listen_Node del TM.

id,Move_PLine(1,2,3,4,5,6)

0:nome connessione
1: una string che definisce il formato: 3 lettere
       1: Motion target format:
        C expressed in Cartesian coordinate
        T: expressed w.r.t. tool coordinate
        J: expressed in joint angles
       2: Speed format:

  ! anche se uso P mi da ok ma non si muove
        A expressed in velocity (mm/s)
       3: Blending format
        P expressed as a percentage
     ! anche se uso R mi da ok ma non si muove

2: coordinate cartesiane relative alla base: x,y,x in mm e Rx Ry Rz
   coordinate relative al tool
3: velocità in % o in mm/s
4: tempo per accelerare a vmax ( Ta) in ms
5: unione traiettorie ( in % ) o in raggio (mm)

%}


%esprimo le coordinate contenute in 'coord' come string e le salvo nella
%stringa 'coo'
coo=strcat('{',string(coord(1)),',',string(coord(2)),',',string(coord(3)),',',string(coord(4)),',',
     string(coord(5)),',',string(coord(6)),'}');

% scrivo il comando, deve risultare in char. La funzione strcat() richiede
% String o Char, per questo trasformo tutto in String
comando=char(strcat('1,Move_PLine("',txt,'",',char(coo),',',string(vel),',',string(Ta),',',string(
     racc),')'));

%Esempio Comando
%comando='1,Move_PLine("TAP",{00,0,40,0,0,0},100,100,0)';

pack=writepack('TMSCT',comando);

```

```
46 fprintf(connessione,'%s',pack);
47 answer=fscanf(connessione);
```

Le variabili di ingresso sono:

- Connessione: oggetto tcpip che rappresenta la connessione tra PC e robot
- txt: è un testo formato da 3 lettere maiuscole che definiscono il formato dei dati successivi e il tipo di movimento:
  1. Formato Coordinate Destinazione: C: per Cartesiano J: per Joints T: per coordinate relative al tool
  2. Formato velocità:

     A: per assoluta in mm/s
  3. Formato unione traiettorie P: Per Percentuale
- coordinate, espresse come vettore in riga $[J_1, J_2, J_3, J_4, J_5, J_6]$ oppure $[X, Y, Z, rx, ry, rz]$
- Velocità in % o in mm/s,le velocità scalano secondo quella che è la velocità di progetto, nel nostro caso dovrebbe essere 100%.
- Tempo di accelerazione in un profilo di velocità trapezoidale, in ms
- unione tra le traiettorie, in % oppure in mm

Come primo passo si trasforma il vettore di coordinate in ingresso in un vettore che formalmente è uguale, ma il tipo di variabile risulta essere string, in modo che si possa concatenare con il resto del comando. Si compone dunque il comando il pacchetto tramite la funzione writepack, lo si invia e si riceve la risposta di conferma. il robot a questo punto dovrebbe muoversi.

Capitolo 4

Prove di Precisione e Ripetibilità su movimenti PTP

4.1 Obiettivi e Procedura

Prima di procedere alla tele-operatività, sono state effettuate prove statistiche per andare a valutare le traiettorie durante il movimento PTP, inoltre sono state stimati i valori di ripetibilità e precisione.

Le prove sono state eseguite utilizzando il movimento PTP, andando a variare velocità e tempo di accelerazione:

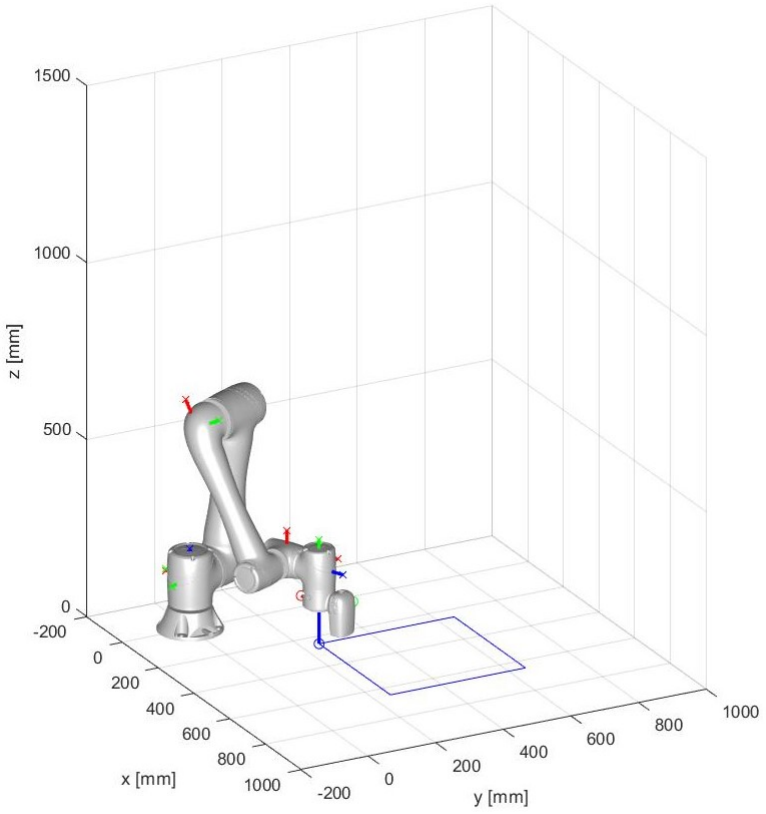
Figura 4.1: Tabella con i parametri delle prove

Vel	10	20	30	40	50	60	70	80	90	100
Ta										
25	ok	ok	ok	ok	ok	ok	X	X	X	X
50						ok	ok	X	X	X
100						ok	ok	X	X	X
250							ok	ok	ok	ok
500							ok	ok	ok	ok

Per la velocità si è valutato tutto il range percentuale ad intervalli di 10%, mentre per i Ta si sono presi 5 valori, da 25 ms, dunque un valore molto basso paragonato al tempo di percorrenza del tratto, a 500 ms che nei casi a velocità elevata rende il profilo di velocità triangolare.

La prova consiste nell'eseguire 100 giri in un quadrato di lato di 400 mm . Il quadrato è di dimensioni elevate in modo da testare il movimento del robot con inerzia elevata.

Figura 4.2: Quadrato con il quale sono state eseguite prove



Come punto di partenza si è scelto $(650, 250, 100)$, da qui partiva il ciclo andando a toccare il punto $(650, 650, 100)$, $(250, 650, 100)$, $(250, 250, 100)$ e infine nuovamente $(650, 250, 100)$.

Si descrive successivamente il file usato per ottenere i dati.

Listing 4.1: File QuadratoContinuo.m per eseguire le prove

```
%% intro
% devo eseguire un quadrato nel piano, in modo da ricavare
%la posizione xyz, quella dei giunti e poi confrontare con varie velocità
%il precorso eseguito e l'errore rispetto alla traiettoria ideale.

%quadrato  650,650  250,250

%% Posizione iniziale
coord=[250,650,100,0,180,0];
y=0;
conf.righty=1;
conf.below=0;
conf.flip=0;
speed=60;
Blending=0;
Ta=25; % millisec
TM.PTP(connessione,'CPP',coord,speed,Ta,Blending,false);
q=TM.ask(connessione,1); %pos iniziale nei giunti
xyz=TM.ask(connessione,2); % posizione iniziale Spazio Operativo.

i=0;
S=[650,650,100,0,180,0;
    650,250,100,0,180,0;
    250,250,100,0,180,0;
    250,650,100,0,180,0];
N=100;
for n=1:N

tic
timerval=0;
risp1=TM.PTP(connessione,'CPP',S(1,:),speed,Ta,Blending,false);
risp2=TM.PTP(connessione,'CPP',S(2,:),speed,Ta,Blending,false);
risp3=TM.PTP(connessione,'CPP',S(3,:),speed,Ta,Blending,false);
risp4=TM.PTP(connessione,'CPP',S(4,:),speed,Ta,Blending,false);

while timerval<4

timerval=toc;
q=AskTM(connessione,1);
c=AskTM(connessione,2);
i=i+1;
Q(i,:)=q;
C(i,:)=c;
v=AskTM(connessione,4);
V(i,:)=v;
end

```

```
48 y=y+1
49
50 end
```

Si definisicono le coordinate del punto di partenza e i parametri di movimento: configurazione, velocità, Ta e raccordo=0. Si porta il robot al punto iniziale tramite un comando PTP e ne si salvano le coordinate nello spazio dei giunti e in quello operativo. Si inizializza un indice $i = 0$ e si crea una matrice S contenente i punti del quadrato in righe. N è il numero di ripetizioni, nel nostro caso 100.

Si inizia dunque il ciclo for per 100 ripetizioni: tramite il comando tic si inizia un timer e vengono dati 4 comandi di movimento corrispondenti ai 4 vertici del quadrato, per i successivi 4 secondi, valore scelto ad occhio in modo tale che il quadrato sia eseguito completamente, vengono continuamente richiesti i valori di coordinate nello spazio dei giunti e cartesiano, per poi essere memorizzati nelle relative matrici C e Q. finiti i 4 secondi un indice y aumenta di valore, questo non ha alcuna funzione all'interno del ciclo ma venendo visualizzata nella command window indicava a che punto dei 100 cicli era arrivata la prova. Finita l'esecuzione era sufficiente salvare le matrici Q e C come file .mat

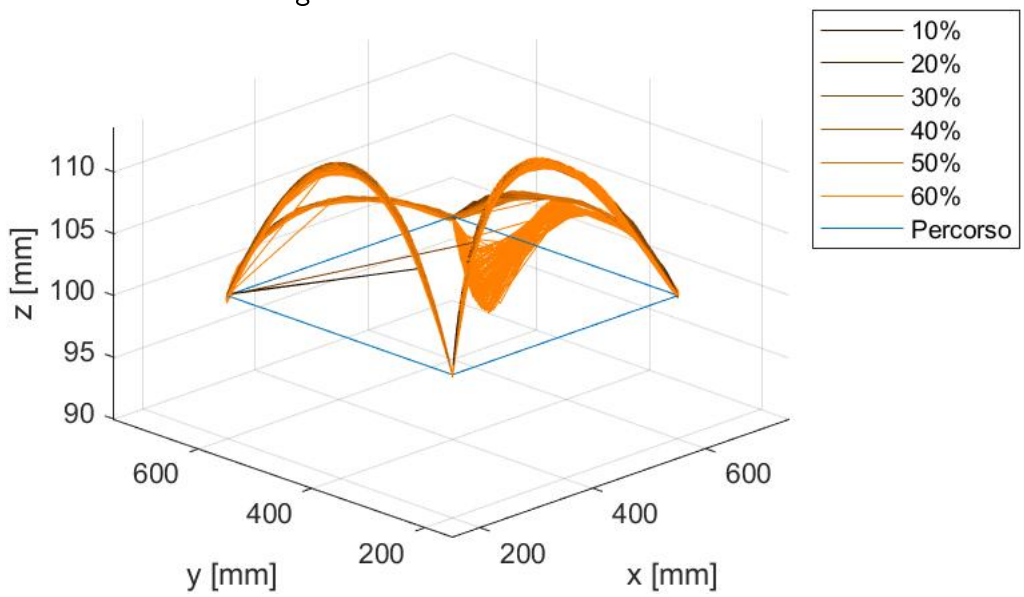
4.2 Prove Continue di Traiettoria

Come prima tipologie di prove, per ogni valore dei parametri velocità e Ta, si eseguono 100 cicli di base, e si misura la posizione più frequentemente possibile in modo da tracciarne la traiettoria. Alcune combinazioni di parametri di movimento sono troppo esigenti e il robot non riesce a esaurirle,andando dunque in errore (solitamente l'errore è del giunto 1 che raggiunge il limite di coppia) e per questo non verranno tracciate, in opposizione se il robot è in grado di eseguire movimenti correttamente in una certa configurazione, si presume che sia in grado di farlo in configurazioni meno esigenti. A questo punto si analizzano le traiettorie al variare della velocità per un valore di Ta.

4.2.1 Ta 25 ms

Questa è la prima prova eseguita, per questo si parte dal 10% di velocità e si aumenta finché possibile.

Figura 4.3: Traiettorie con Ta 25 ms



Le traiettorie a 10%-50% di velocità sono molto simili, si vede un leggero abbassamento della quota Z all'aumentare della velocità nel tratti da un punto da un altro.

La grande differenza si nota a velocità 60%: nel momento in cui il robot passa da $(250, 650, 100)$ a $(650, 650, 100)$, si ferma bruscamente causando una oscillazione che non riesce ad essere smorzata prima che il robot riparta per il punto successivo, dunque la brusca variazione di traiettoria è proprio dovuta all'oscillazione residua che viene distribuita lungo il movimento successivo.

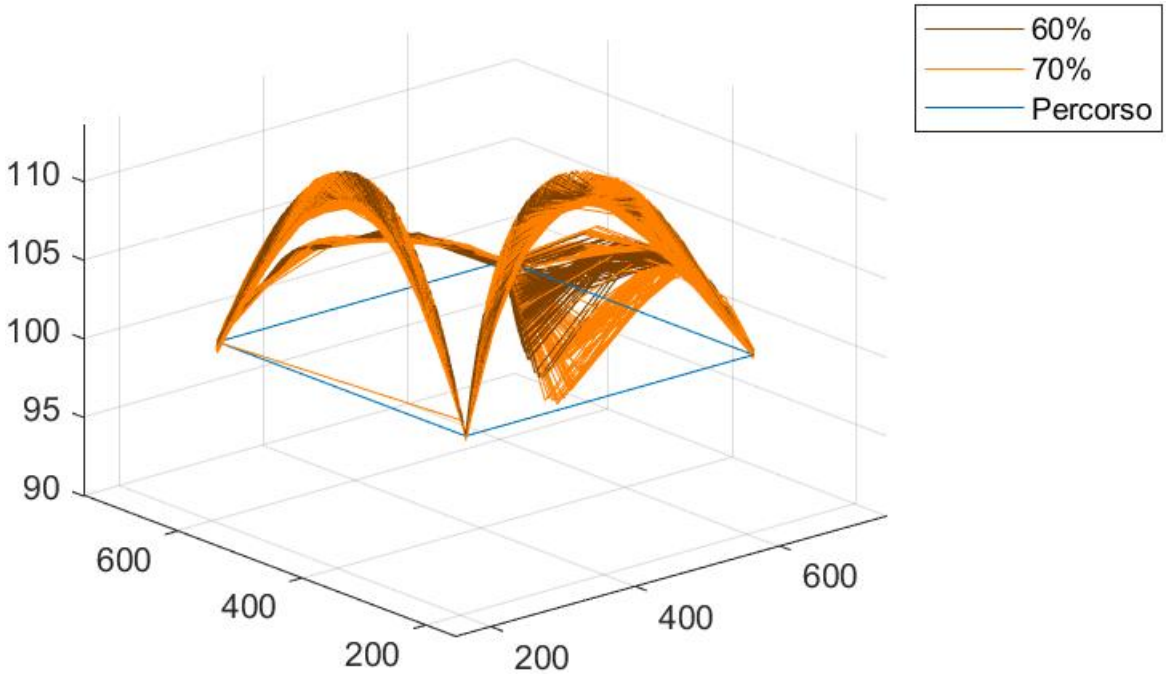
La prova viene interrotta al 70% perché dal punto (650,650,100) al successivo, il giunto 1 deve dare un forte scatto per raggiungere la velocità, ma il robot è quasi del tutto esteso, dunque con inerzia elevata, e il giunto 1 raggiunge il limite massimo di coppia esercitata.

Questa e esperienza suggerisce di valutare il cambio di traiettoria e la fattibilità del percorso variando il tempo di accelerazione.

4.2.2 Ta 50 ms

questa prova parte dal 60%,poiché per velocità inferiori e Ta più esigenti non si sono verificate anomalie.

Figura 4.4: Traiettorie con Ta 50 ms

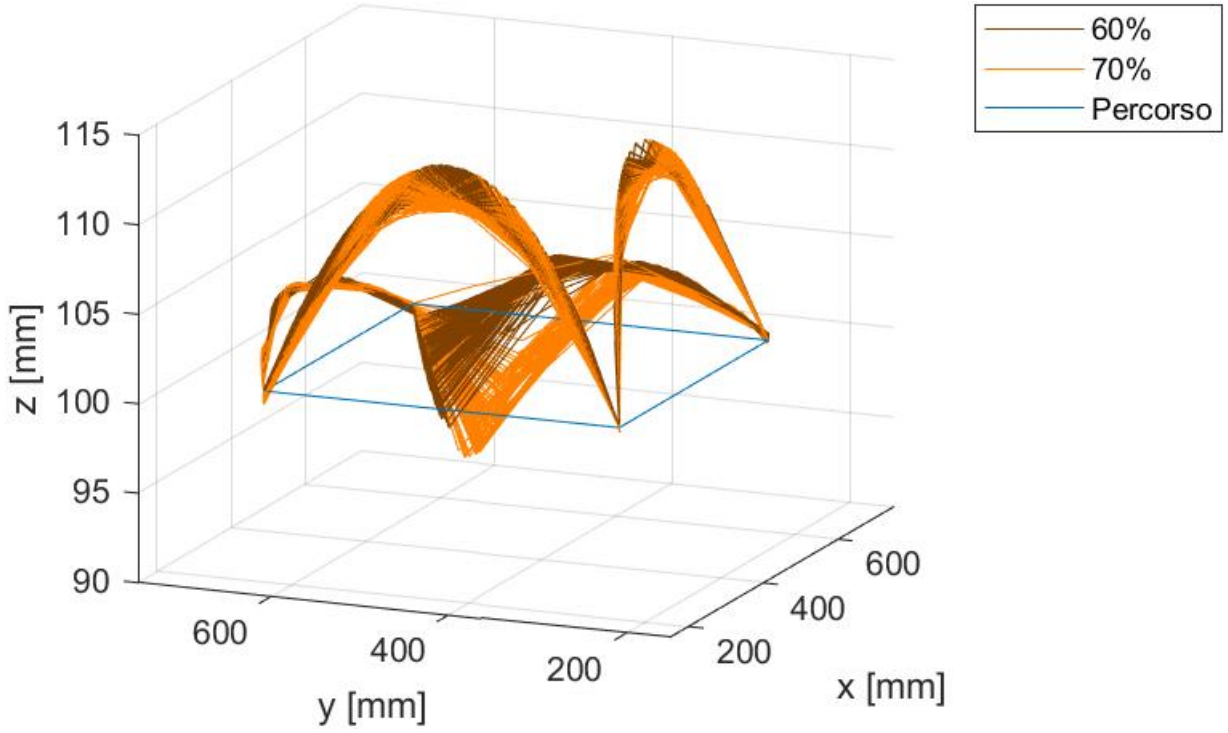


In questo caso le traiettorie sono sempre disturbate dall'oscillazione, tuttavia un aumento di Ta permette al giunto 1 di muovere il robot al 70% di velocità.

4.2.3 Ta 100 ms

questa prova parte dal 60%, tuttavia comprende solo 60%-70% perché anche con 100 ms raggiunge il limite di coppia.

Figura 4.5: Traiettorie con Ta 100 ms



In questo caso le traiettorie sono sempre disturbate dall'oscillazione.

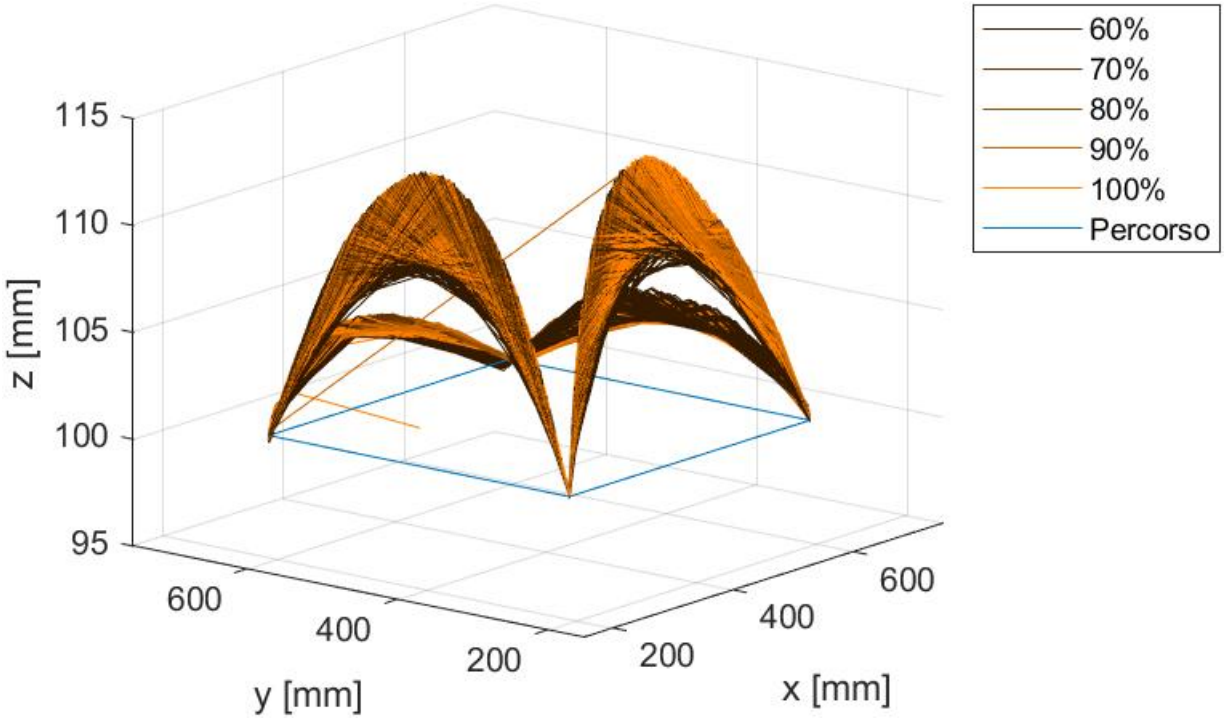
Vedendo che non c'è stato alcun cambiamento il prossimo Ta sarà 250ms

4.2.4 Ta 250 ms

questa prova parte dal 60% e raggiunge il 100% di velocità. Le traiettorie disegnate nel grafico risultano più frastagliate a causa della bassa frequenza di campionamento, che non aumenta al variare della velocità.

Le linee che non sembrano congiungersi con le traiettorie sono causate da errori durante l'acquisizione dei dati o da problemi durante l'acquisizione e dunque dalla unione di diversi file C senza però continuità di posizione.

Figura 4.6: Traiettorie con Ta 250 ms

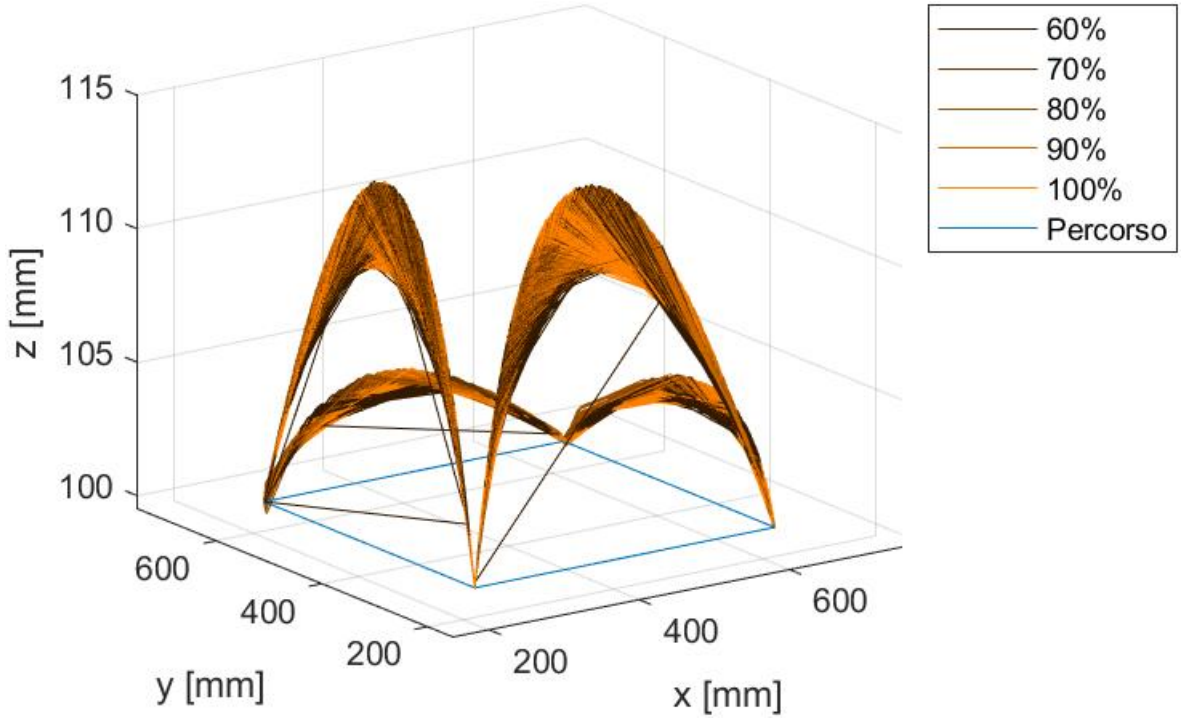


In questo caso posso finalmente raggiungere la velocità massima e anche le traiettorie non sono troppo disturbate dall'oscillazione.

4.2.5 Ta 500 ms

questa prova parte dal 60% e raggiunge il 100% di velocità. Le traiettorie disegnate nel grafico risultano più frastagliate a causa della bassa frequenza di campionamento, che non aumenta al variare della velocità.

Figura 4.7: Traiettorie con Ta 500 ms



In questo caso posso finalmente raggiungere la velocità massima e anche le traiettorie non sono troppo disturbate dall'oscillazione.

Le linee che non sembrano congiungersi con le traiettorie sono causate da errori durante l'acquisizione dei dati o da problemi durante l'acquisizione e dunque dalla unione di diversi file C senza però continuità di posizione.

4.3 Prove su movimenti PTP

La seconda tipologia di prove è simile al precedente, tuttavia il percorso quadrato anziché essere continuo, è interrotto in ogni vertice, in modo che il robot abbia tempo di assestarsi. Le prove sono state eseguite variando il Ta e la velocità con la stessa configurazione della prima tipologia, ogni combinazione ha un totale di 100 esecuzioni, dunque 400 punti totali, 100 per ogni vertice.

Figura 4.8: Combinazini della seconda tipologia di prove

Vel	10	20	30	40	50	60	70	80	90	100
Ta										
25	ok	ok	ok	ok	ok	ok	ok	X	X	X
50	ok	ok	ok	ok	ok	ok	ok	X	X	X
100	ok	ok	ok	ok	ok	ok	ok	ok	X	X
250	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok
500	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok

Per ogni valore di Ta vengono presentate le nuvole di punti attorno ai vertici, viene calcolata la precisione assoluta e viene calcolata la ripetibilità del movimento al variare della velocità. Per la precisione assoluta si calcola la distanza spaziale dal punto reale al vertice del quadrato e infine si calcola la media aritmetica per ogni vertice. Supponendo il modello gaussiano come adatto al sistema, si calcola lo scarto quadratico medio, la ripetibilità si considera $3 * \sigma$, con una probabililità di 99,8 %.

Si descrive successivamente il file utilizzato per la raccolta dei dati:

Listing 4.2: File QuadratoVertici.m per eseguire le prove

```
%% intro
% devo eseguire un quadrato nel piano, in modo da ricavare
%la posizione xyz, quella dei giunti  AI VERTICI  e poi confrontare con varie velocità
%il precorso eseguito e l'errore rispetto alla traiettoria ideale.

%quadrato  650,650  250,250

%% Posizione iniziale
coord=[250,650,100,0,180,0];
y=0;
```

```
conf.righty=1;
conf.below=0;
conf.flip=0;
speed=60;
Blending=0;
Ta=25; % millisec
TM.PTP(connessione,'CPP',coord,speed,Ta,Blending,false,conf);
q=TM.ask(connessione,1); %pos iniziale nei giunti
xyz=TM.ask(connessione,2); % posizione iniziale Spazio Operativo.

i=0;
S=[650,650,100,0,180,0;
    650,250,100,0,180,0;
    250,250,100,0,180,0;
    250,650,100,0,180,0];
N=2;
for n=1:N


for j = 1:4


risp1=TM.PTP(connessione,'CPP',S(j,:),speed,Ta,Blending,false,conf);
n
pause(1.8)

q=AskTM(connessione,1);
c=AskTM(connessione,2);
i=i+1;
Q(i,:)=q;
C(i,:)=c;
v=AskTM(connessione,4);
V(i,:)=v;

end


end
```

Come prima azione viene definito il punto iniziale, viene inizializzato l'indice y e viene definita la configurazione assieme ai parametri di movimento.

Si fa poi muovere il robot al punto iniziale e ne si chiedono le coordinate nello spazio dei giunti e in quello cartesiano. Viene definita la matrice contenente le coordinate dei vertici del quadrato e dunque si inizia il ciclo di prove. N indica il numero di prove, in questo caso 100. Ogni iterazione del ciclo corrisponde

all'esecuzione di un ulteriore ciclo le cui azioni sono: muoversi al punto successivo, aspettare che ci si assesti, richiedere le coordinate e memorizzarle nelle matrici Q e C; questo per ogni lato del quadrato.

Finita la prova non resta che salvare le matrici Q e C come file .mat .

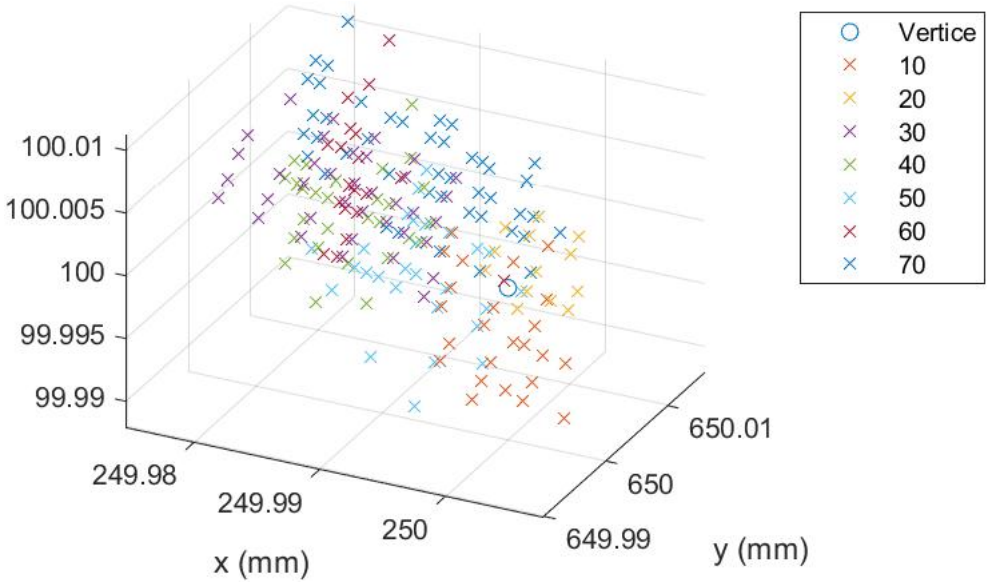
Nei grafici successivi i punti verranno rappresentati coi seguenti colori:

Vertice 250 650	blu
Vertice 650 650	azzurro
Vertice 650 250	rosso
Vertice 250 250	Verde

4.3.1 Ta 25 ms

Si presentano le distribuzioni di punti attorno ai vertici del quadrato:

Figura 4.9: Vertice (250,650)



è facile notare che sembrano quasi allineati in una griglia più che sparsi a caso nello spazio, questo è semplicemente la discretizzazione della posizione dei rotori che si riflette nel movimento.

Si riporta dunque l'andamento dell'errore di posizionamento rispetto al vertice:

Figura 4.10: Vertice (650,650)

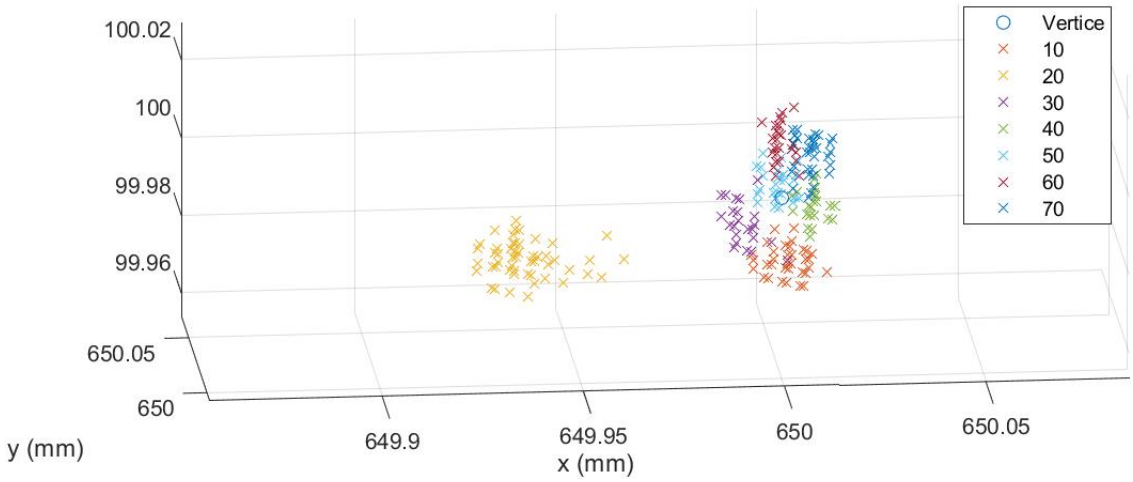


Figura 4.11: Vertice (250,250)

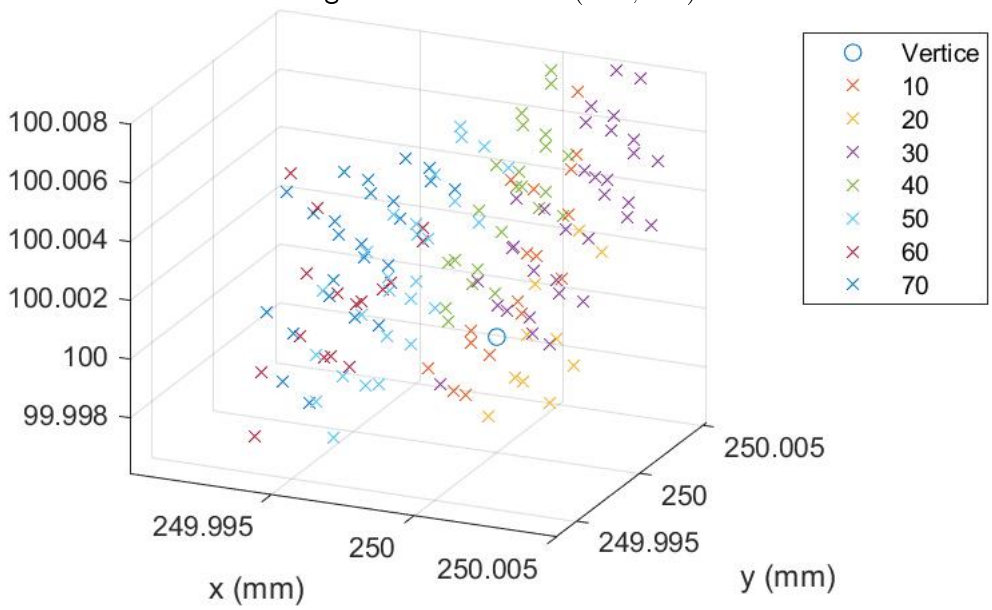
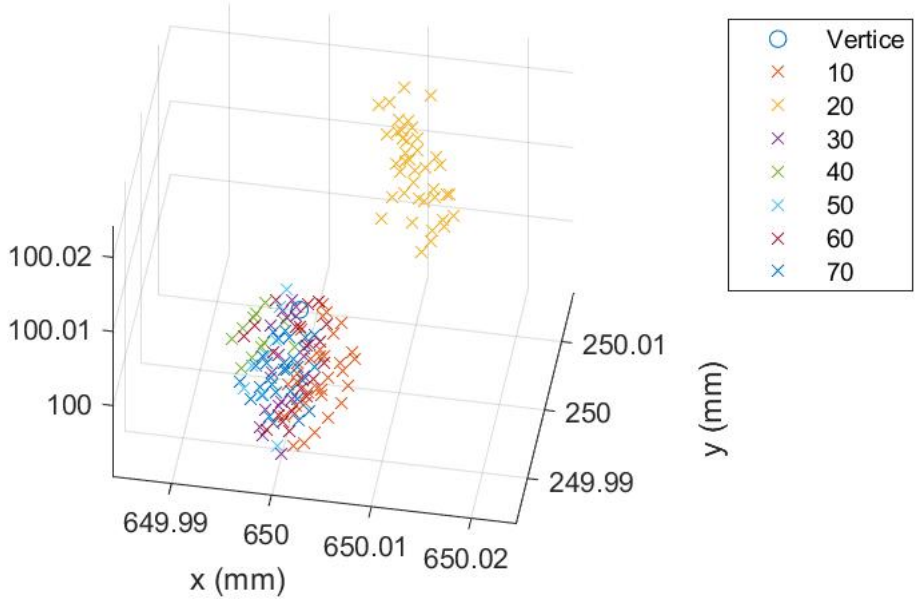


Figura 4.12: Vertice (650,250)

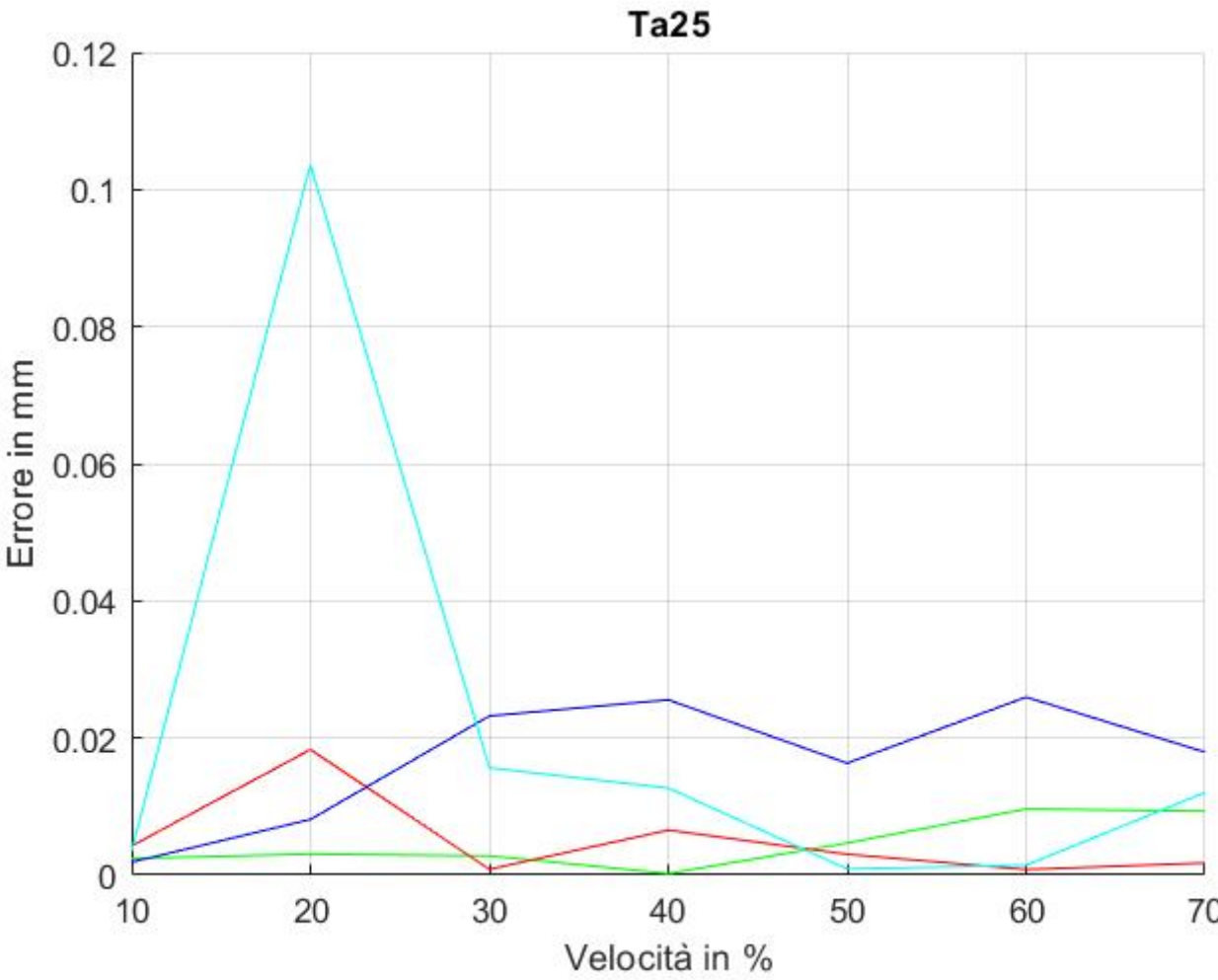


Ogni linea nel grafico rappresenta un vertice del quadrato, subito si nota il picco del vertice (650,650) al 20%, anche guardando la nube di punti si evidenzia un errore elevato, probabilmente causato da fenomeni specifici per questa configurazione.

In generale si può evidenziare come al punto 3 (250,650), che è la fine del tratto con partenza dal punto 2 (650,650), si possa notare una sensibile dipendenza dalla velocità.

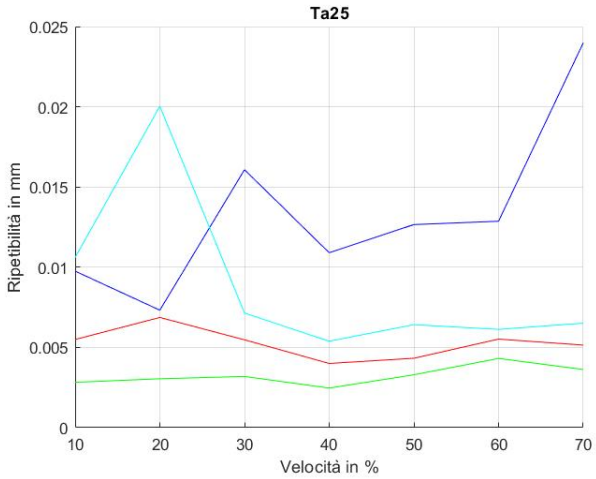
Con questi valori non sembra essere evidente una dipendenza dovuta alla lontananza media dalla base, come sarà visibile poi con altri tempi di accelerazione. Ad esclusione del picco di 0.1 mm, l'errore rimane sempre inferiore ai 0.03mm.

Figura 4.13: Andamento dell'errore assoluto al variare della velocità (Ta=25ms)



La ripetibilità viene mostrata in uno schema simile:

Figura 4.14: Andamento della ripetibilità al variare della velocità (Ta=25ms)



In questo caso è presente sia una dipendenza dalle velocità, sia rispetto alla distanza dalla base. Il valore più basso lo si riscontra infatti al punto 4 (250,250) poi il punto 1 (650,250) e infine il punto 2 (650,650). La ripetibilità al punto 3 invece presenta una forte dipendenza rispetto alla velocità, per le stesse considerazioni sull'errore generale.

4.3.2 Ta 50 ms

Si presentano le distribuzioni di punti attorno ai vertici del quadrato:

Figura 4.15: Vertice (250,650)

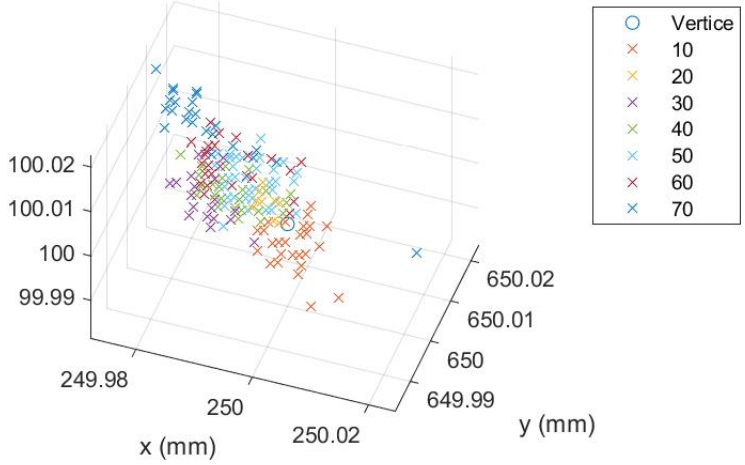


Figura 4.16: Vertice (650,650)

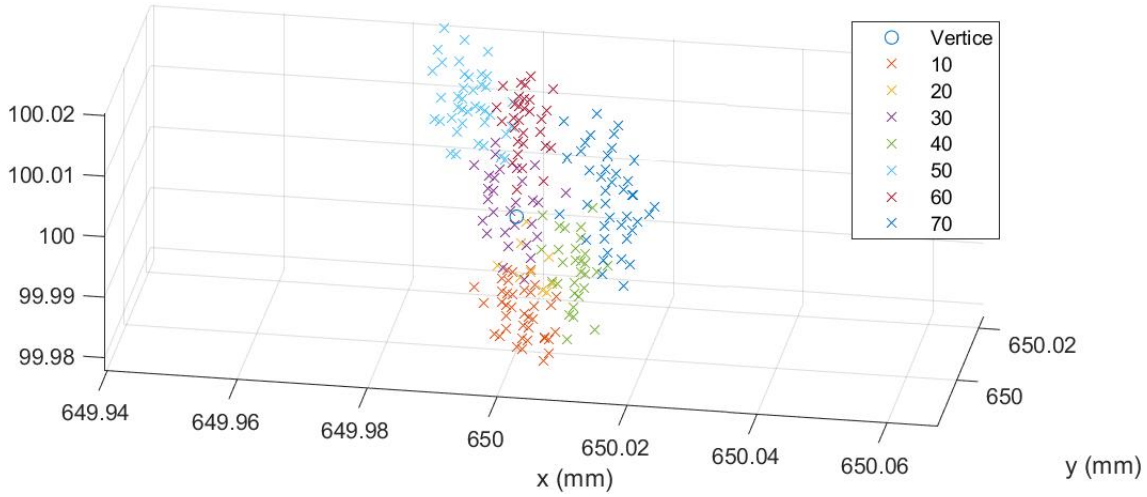


Figura 4.17: Vertice (250,250)

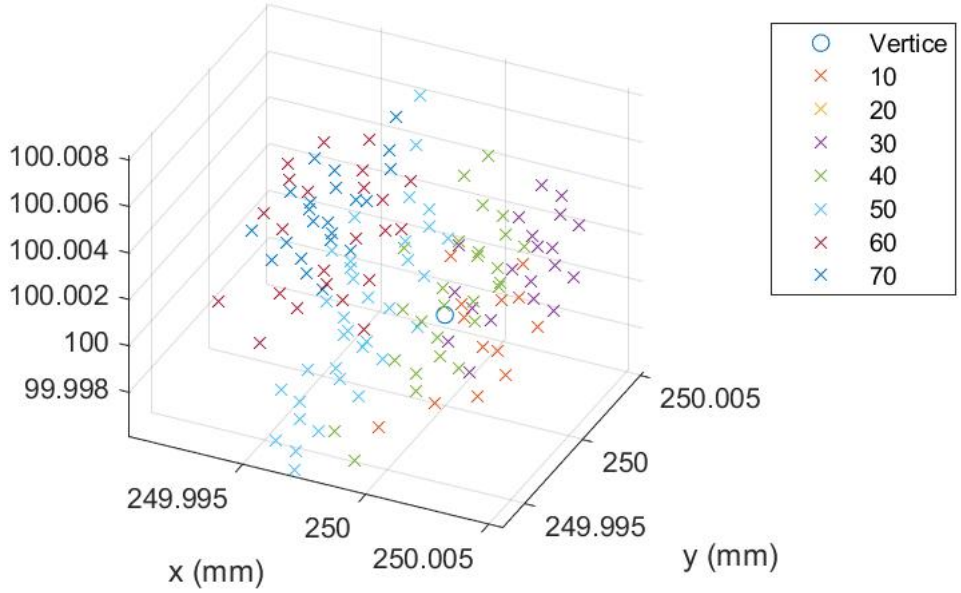
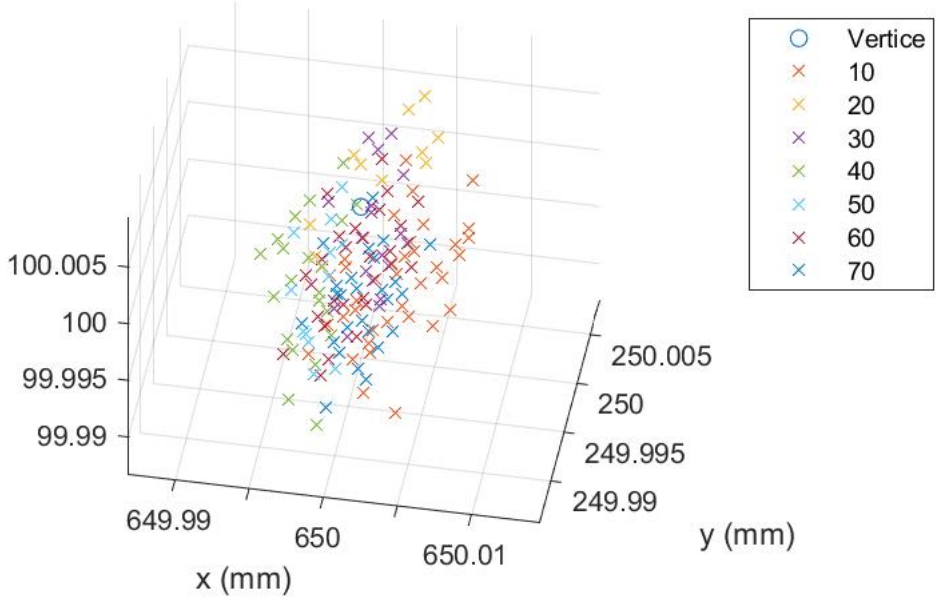


Figura 4.18: Vertice (650,250)



Anche con $Ta = 50ms$ sono evidenti distribuzioni a griglia, dovuta alla discretizzazione dei motori del robot.

Si riporta dunque l'andamento dell'errore di posizionamento rispetto al vertice:

In questo caso non è possibile dire se sia presente o meno una dipendenza dalla distanza dalla base, perché sono presenti sbalzi sui valori, è comunque evidente come al punto 2 e 3 questi fenomeni siano più importanti. I valori sono tutti inferiori a 0.035 mm, inizia anche a vedersi un miglioramento generale rispetto alle misurazioni con Ta 25 ms.

La ripetibilità viene mostrata in uno schema simile:

In questo caso è presente sia una dipendenza dalle velocità, sia rispetto alla distanza dalla base. Il valori più bassi si riscontrano infatti al punto 4 (250,250) e al punto 1 (650,250), mentre ai punti 2 e 3 sono maggiori. La ripetibilità al punto 3 invece presenta una forte dipendenza rispetto alla velocità, per le stesse considerazioni sull'errore generale.

Si nota un miglioramento rispetto al tempo Ta=25 ms

Figura 4.19: Andamento dell'errore assoluto al variare della velocità (Ta=50ms)

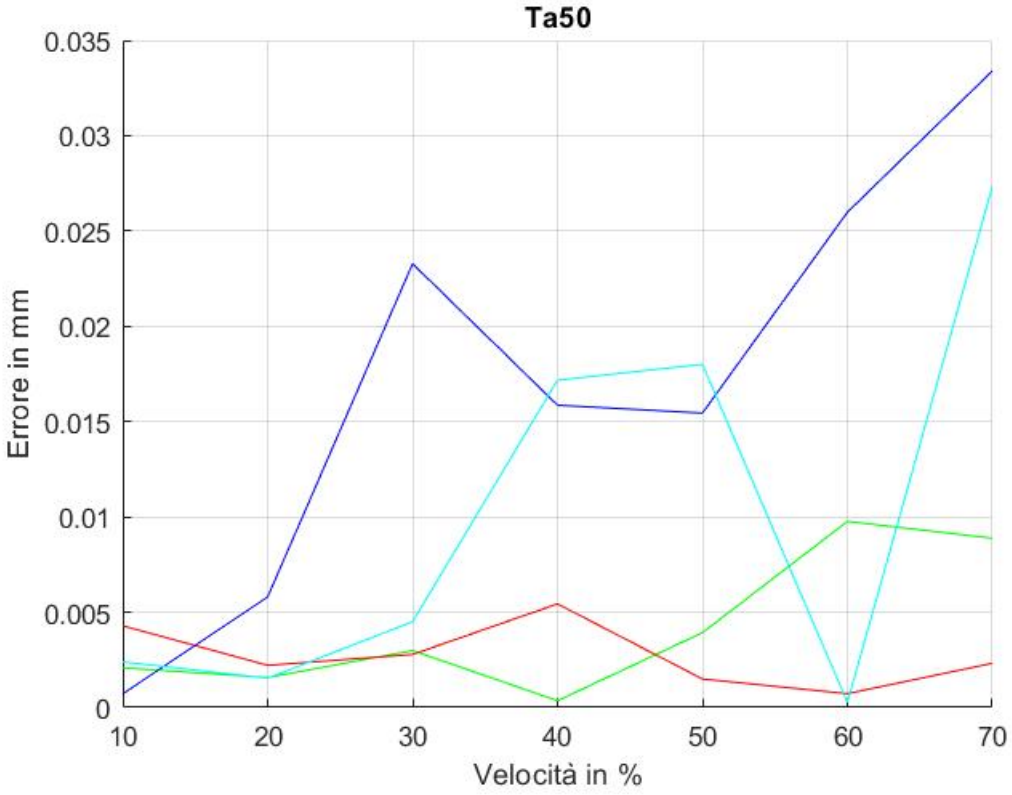
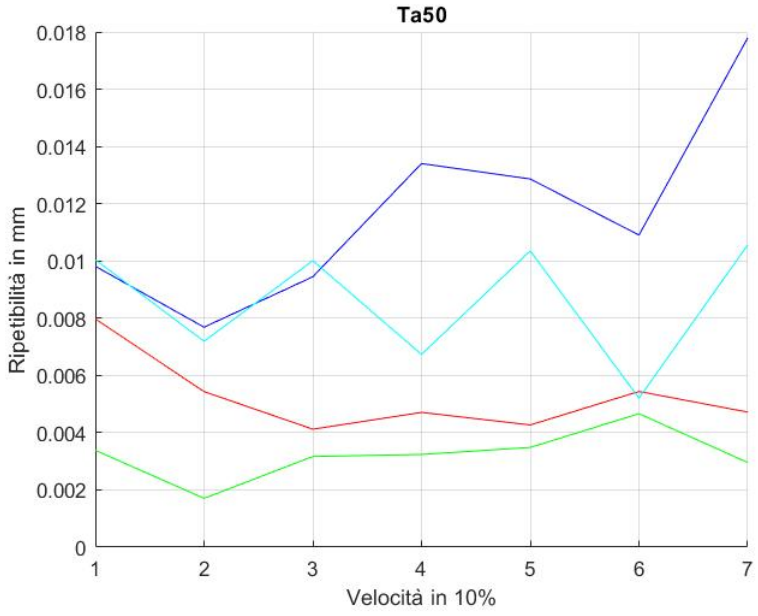


Figura 4.20: Andamento della ripetibilità al variare della velocità (Ta=50ms)



4.3.3 Ta 100 ms

Si presentano le distribuzioni di punti attorno ai vertici del quadrato:

Figura 4.21: Vertice (250,650)

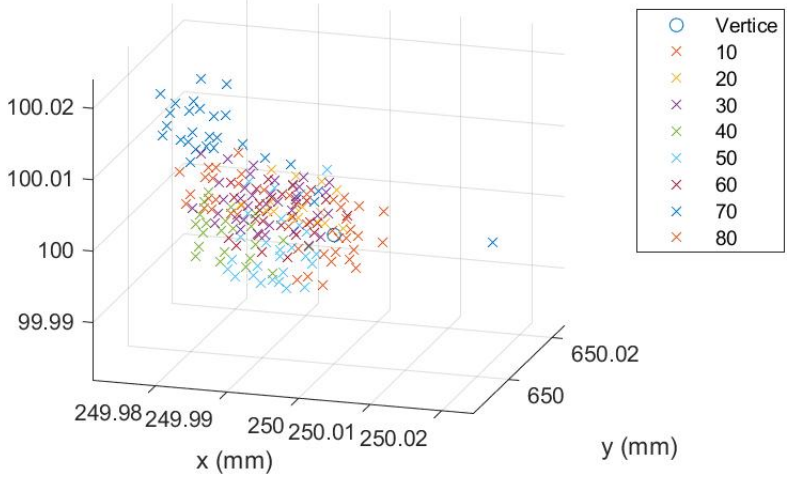


Figura 4.22: Vertice (650,650)

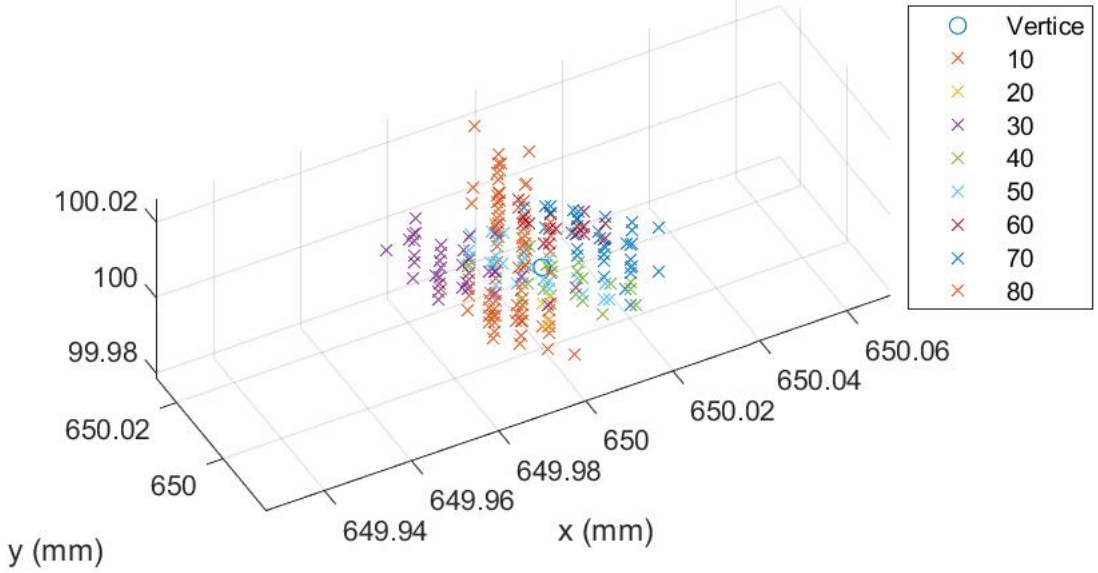


Figura 4.23: Vertice (250,250)

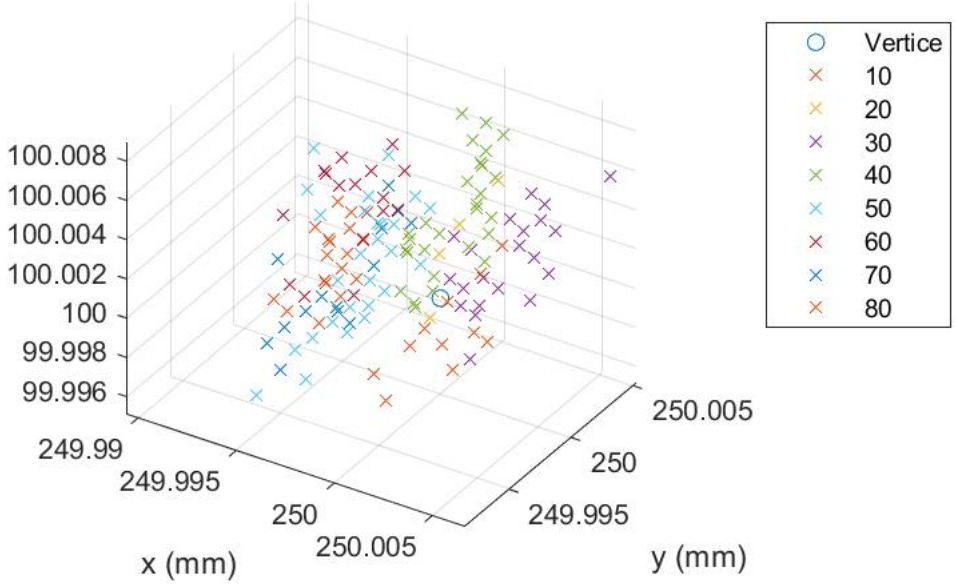
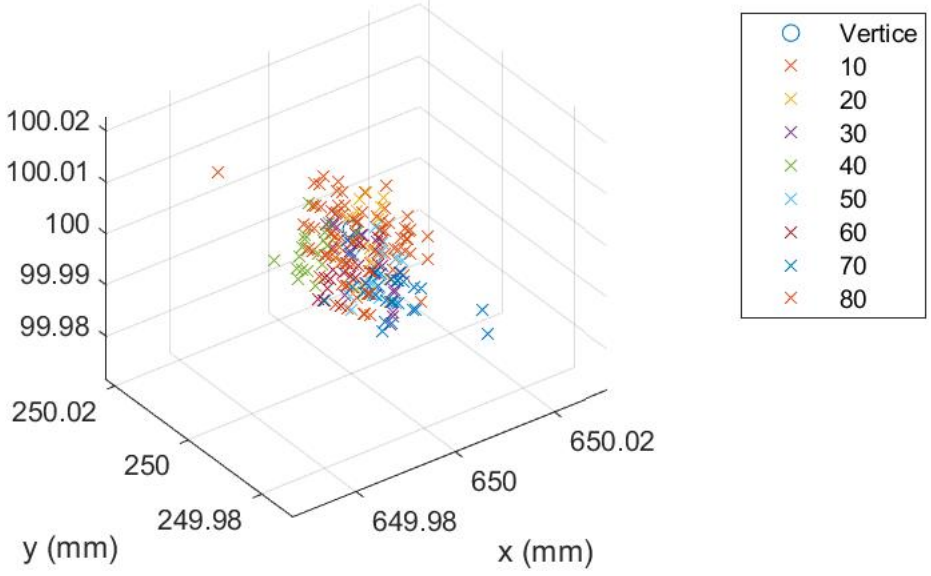


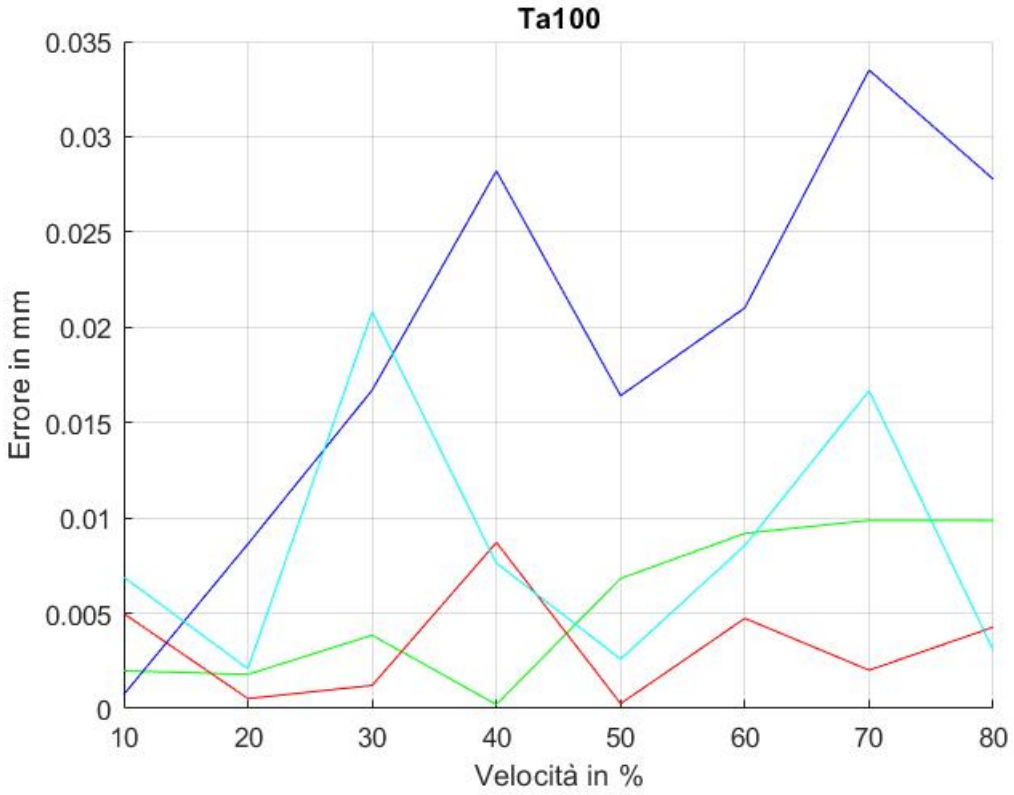
Figura 4.24: Vertice (650,250)



Anche con $Ta = 100ms$ sono evidenti distribuzioni a griglia, dovuta alla discretizzazione dei motori del robot.

Si riporta dunque l'andamento dell'errore di posizionamento rispetto al vertice:

Figura 4.25: Andamento dell'errore assoluto al variare della velocità (Ta=100ms)

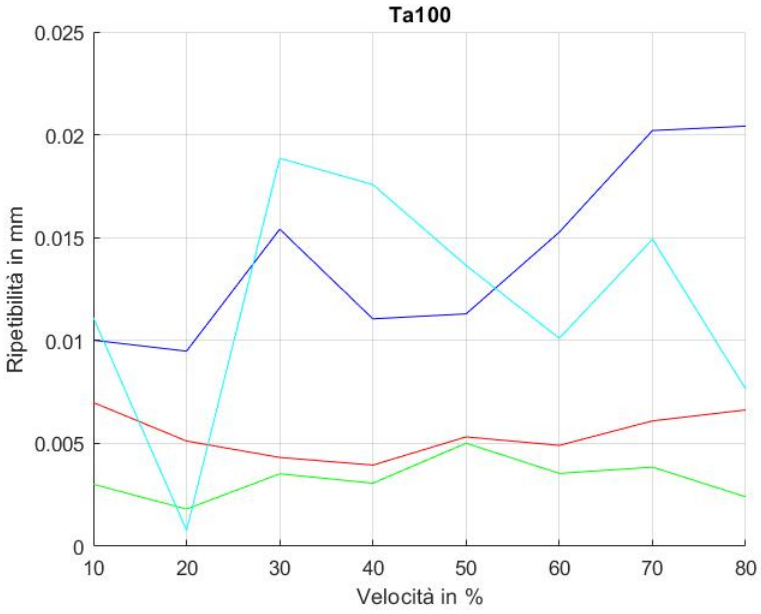


In questo caso è presente sia una dipendenza dalle velocità, sia rispetto alla distanza dalla base. I vertici 4 ed 1 hanno errore più basso mentre i vertici 2 e 3 più elevati. Questo però hanno elevate variazioni, i valori sono in miglioramento rispetto al caso precedente.

La ripetibilità viene mostrata in uno schema simile.

In questo caso è presente sia una dipendenza dalle velocità, sia rispetto alla distanza dalla base. Il valori più bassi si riscontrano infatti al punto 4 (250,250) e al punto 1 (650,250), mentre ai punti 2 e 3 sono maggiori. La ripetibilità al punto 3 invece presenta una forte dipendenza rispetto alla velocità, per le stesse considerazioni sull'errore generale.

Figura 4.26: Andamento della ripetibilità al variare della velocità (Ta=100ms)

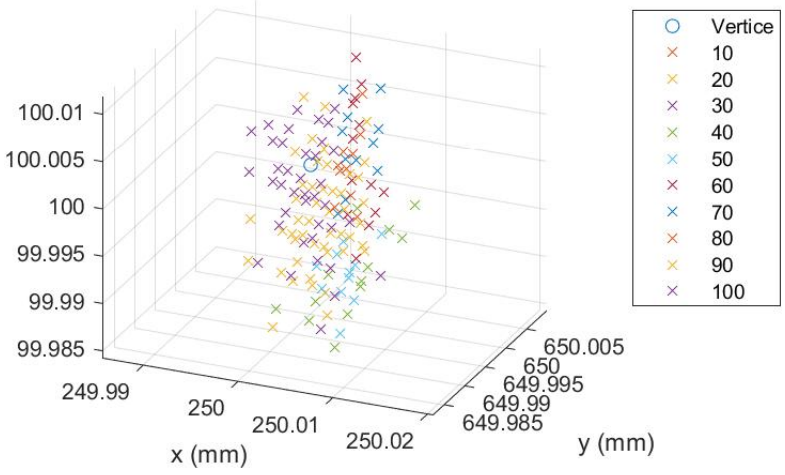


Non si nota un evidente miglioramento rispetto al tempo Ta=50 ms

4.3.4 Ta 250 ms

Si presentano le distribuzioni di punti attorno ai vertici del quadrato:

Figura 4.27: Vertice (250,650)



Anche con $Ta = 250ms$ sono evidenti distribuzioni a griglia, dovuta alla discretizzazione dei motori del robot.

Figura 4.28: Vertice (650,650)

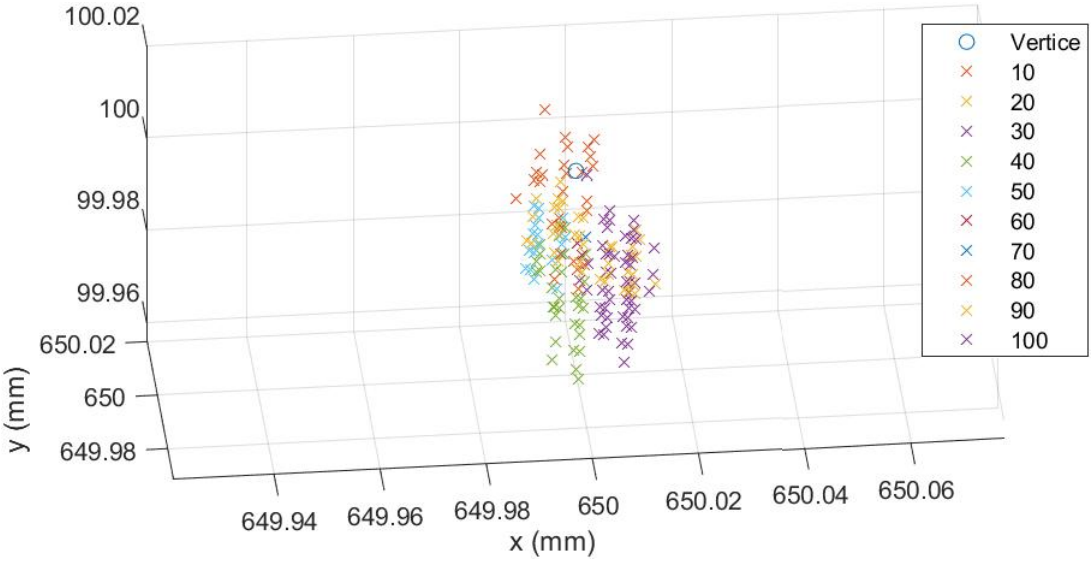


Figura 4.29: Vertice (250,250)

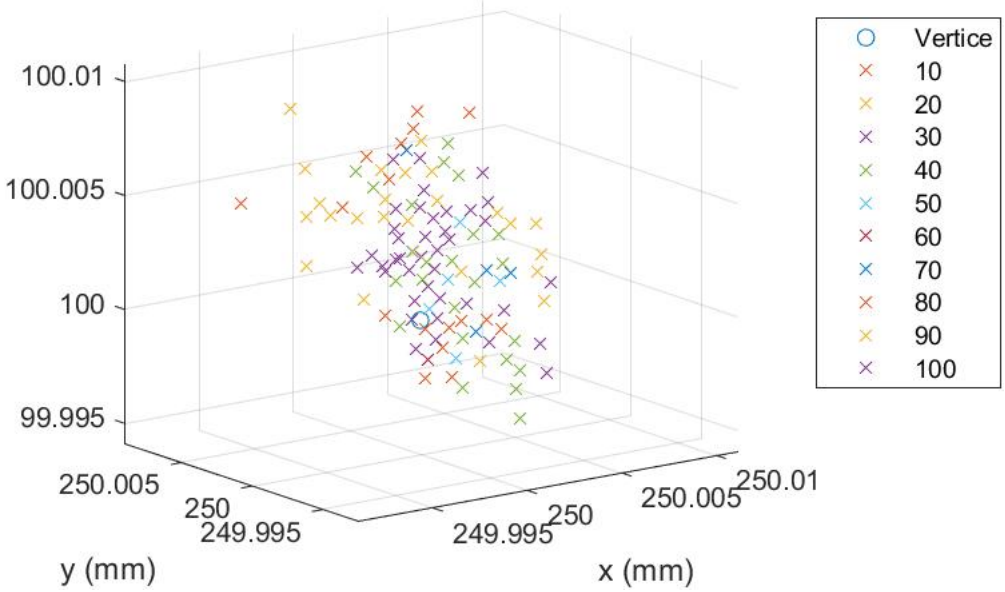
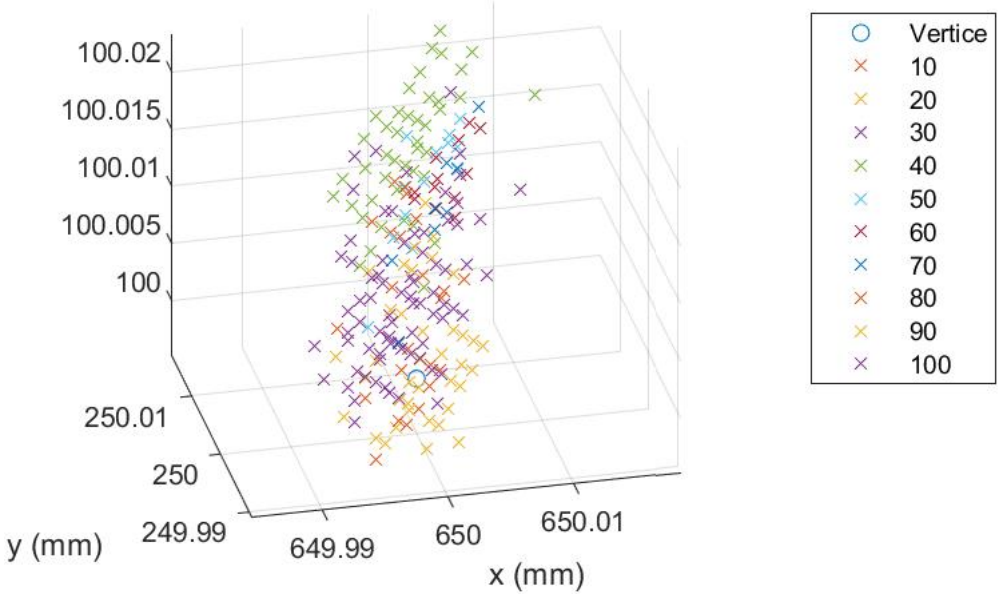


Figura 4.30: Vertice (650,250)



Si riporta dunque l'andamento dell'errore di posizionamento rispetto al vertice:

Subito a 250 ms come tempo di accelerazione si toglie una dipendenza all'errore rispetto alla velocità, rimane dunque solo la presenza di un disturbo costante e la dipendenza dalla distanza dalla base robot: come in precedenza i vertici 4 ed 1 hanno errore più basso mentre i vertici 2 e 3 più elevati.

Si nota un miglioramento dell'errore massimo.

La ripetibilità viene mostrata in uno schema simile.

in questo caso il disturbo di fondo porta a mischiarsi dei valori dei vari vertici senza evidenziare alcuna dipendenza.

4.3.5 Ta 500 ms

Si presentano le distribuzioni di punti attorno ai vertici del quadrato:

Anche con $Ta = 500ms$ sono evidenti distribuzioni a griglia, dovuta alla discretizzazione dei motori del robot.

Si riporta dunque l'andamento dell'errore di posizionamento rispetto al vertice:

Questo caso leggermente più pulito rispetto al precedente.

Figura 4.31: Andamento dell'errore assoluto al variare della velocità (Ta=250ms)

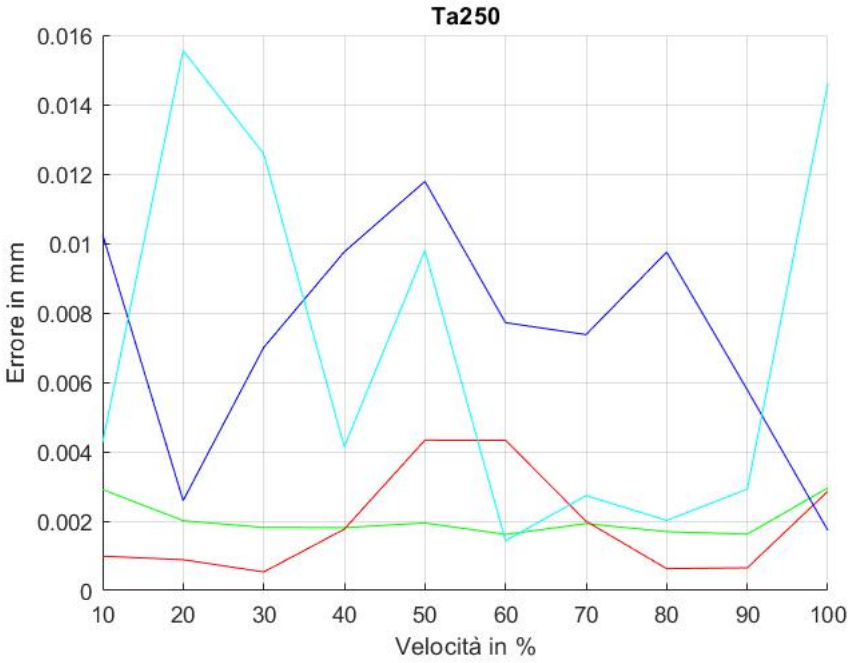


Figura 4.32: Andamento della ripetibilità al variare della velocità (Ta=250ms)

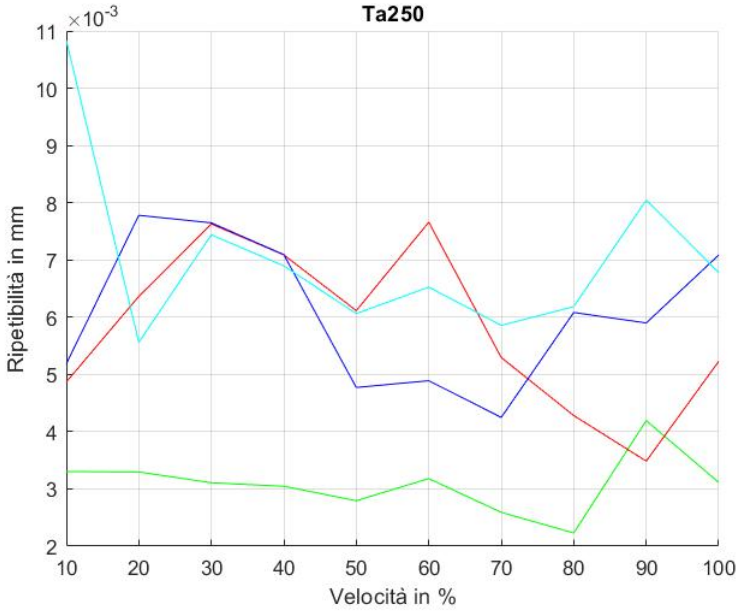


Figura 4.33: Vertice (250,650)

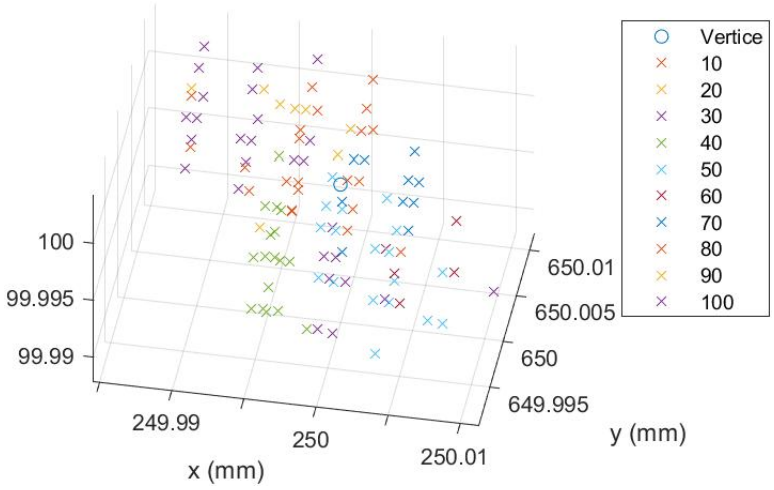


Figura 4.34: Vertice (650,650)

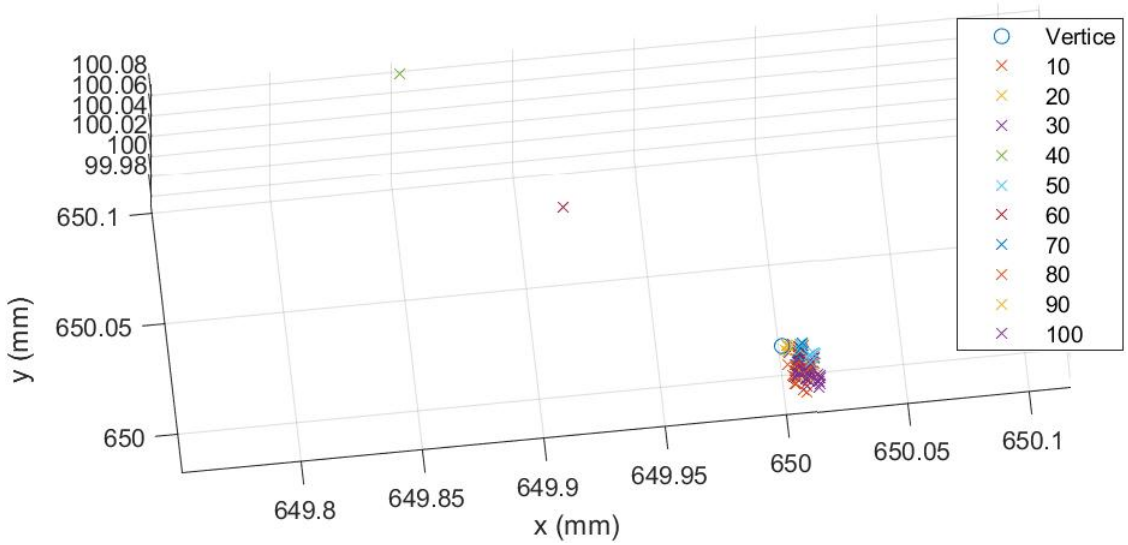


Figura 4.35: Vertice (250,250)

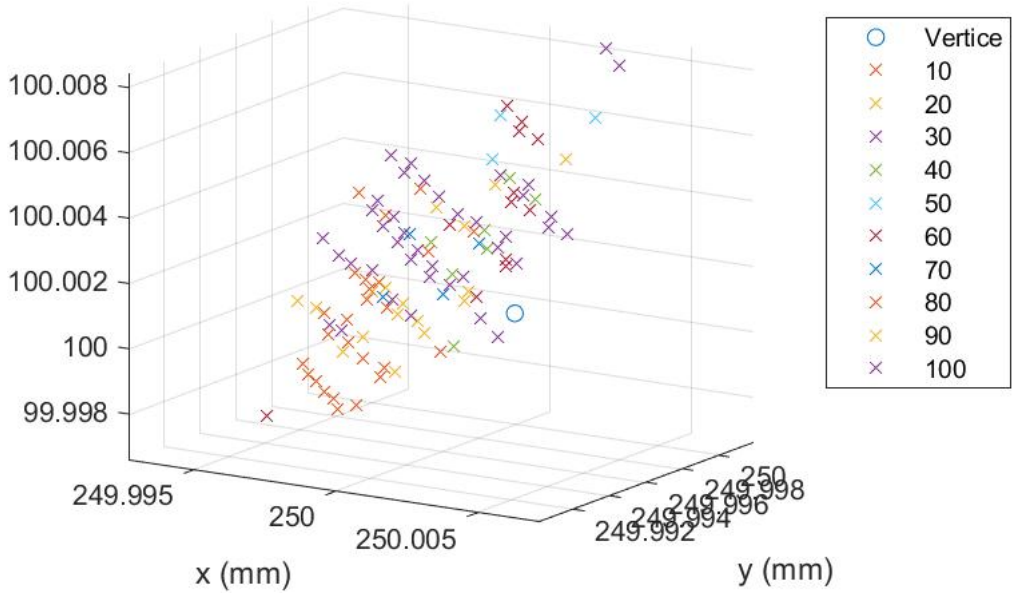


Figura 4.36: Vertice (650,250)

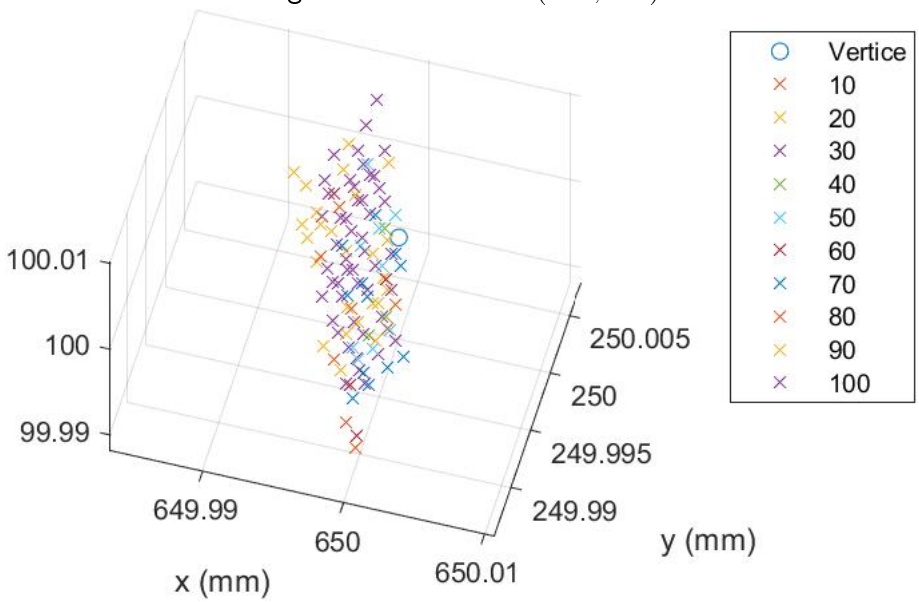
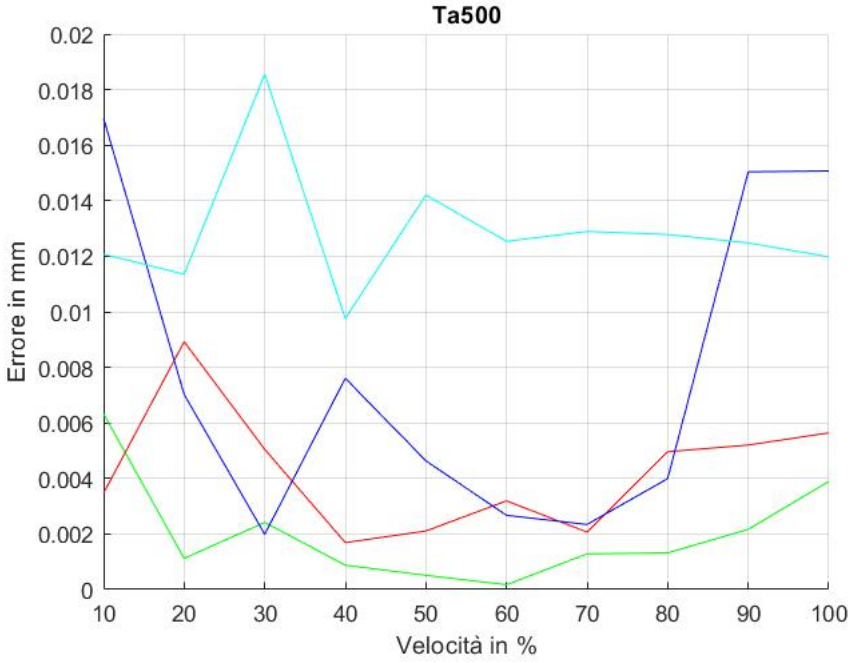


Figura 4.37: Andamento dell'errore assoluto al variare della velocità (Ta=500ms)

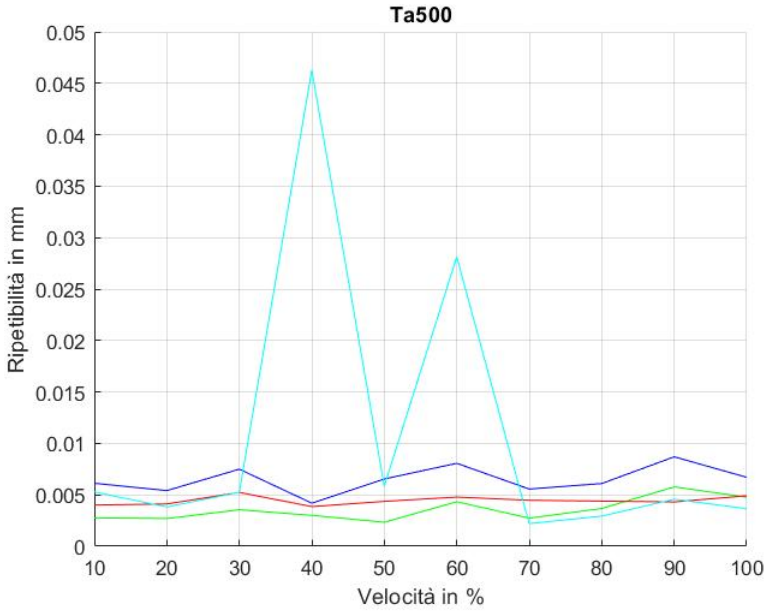


Si nota un miglioramento dell'errore massimo.

La ripetibilità viene mostrata in uno schema simile.

Questo schema, ad esclusione dei due picchi al punto 2, é possibile mostrare che la ripetibilità non è più dipendente dalla distanza dalla base ma si assesta su un valore di 0.005 mm.

Figura 4.38: Andamento della ripetibilità al variare della velocità (Ta=500ms)



4.4 Considerazioni

Le prove finora considerate non sono sufficienti per avere chiaro il comportamento del robot, tuttavia forniscono una buona base di partenza su che cosa sia necessario approfondire ulteriormente.

Considerando la tipologia di prove è evidente che, per ottenere una traiettoria accettabile, sia necessario trovare un equilibrio tra Ta e velocità rispetto alla posizione che il robot dovrà raggiungere. In ogni caso un ottimo indicatore è sentire se i freni si attivino durante il movimento:nei tratti più esigenti per mantenere velocità e Ta il robot azionava i freni, andando a rallentare non coi motori ma meccanicamente. Questo oltre ad aumentare il rumore durante il funzionamento è indice che il robot è sotto dura prova, dunque è bene considerare soluzioni diverse.

Inoltre, è necessario approfondire il comportamento con altri percorsi: il problema si evidenzia quando si lavora lontano dalla base e muovendosi lungo gli assi della terna di riferimento, sarebbe utile provare a lavorare cercando di focalizzare l'attenzione più sul giunto 1 e 2 che essendo quelli più sollecitati sono la causa della variazione. Anche percorsi disomogenei che evidenziano differenze tra il movimento di allontanamento e quello di avvicinamento alla base.

Passando invece alla seconda tipologia di prove, che è meno qualitativa e più quantitativa, vengono presentati gli schemi precedenti uniti, in aggiunta si è disegnato uno scheletro di quella che è la ripetibilità dichiarata al costruttore: $\pm 0.05mm$.

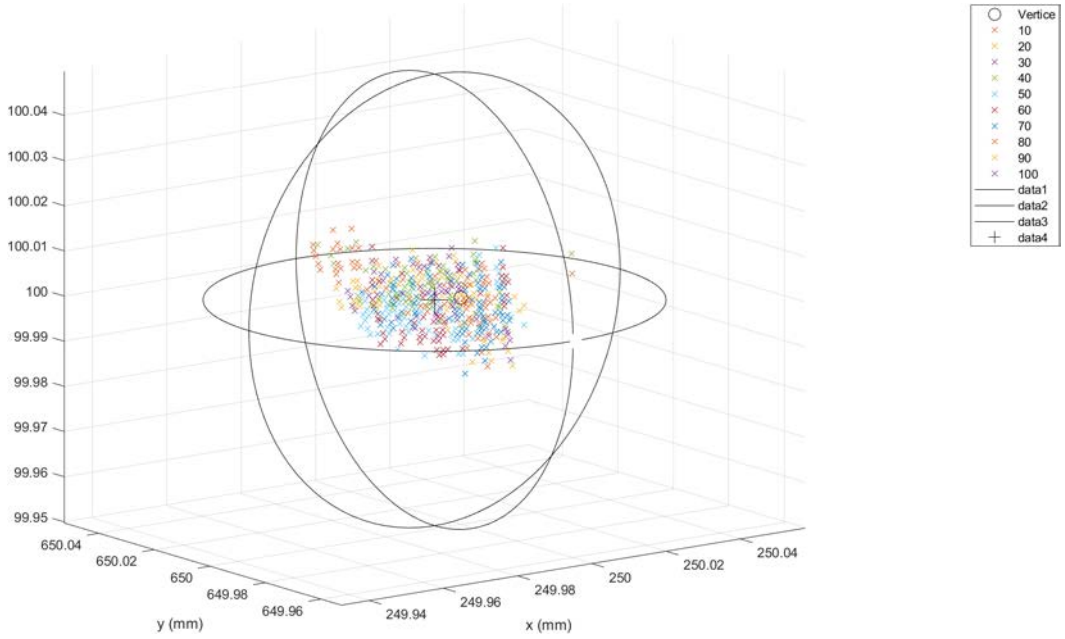


Figura 4.39: Vertice 250 650 con tutti i valori di Ta

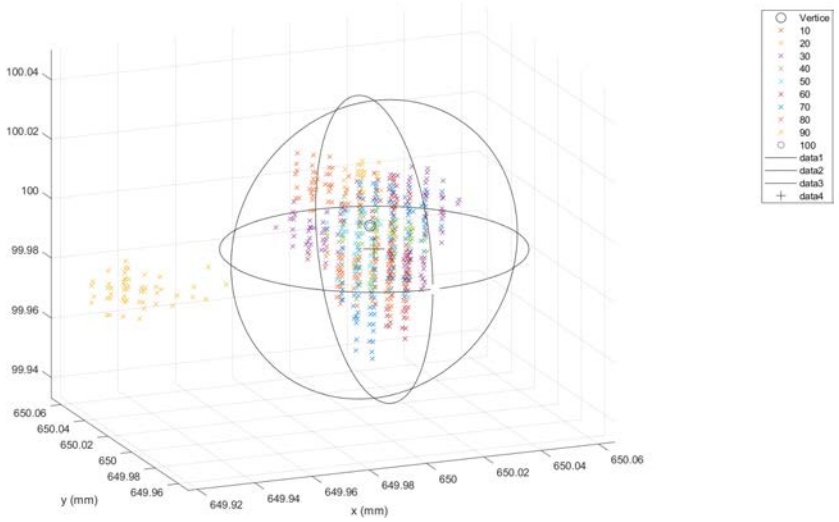


Figura 4.40: Vertice 650 650 con tutti i valori di Ta

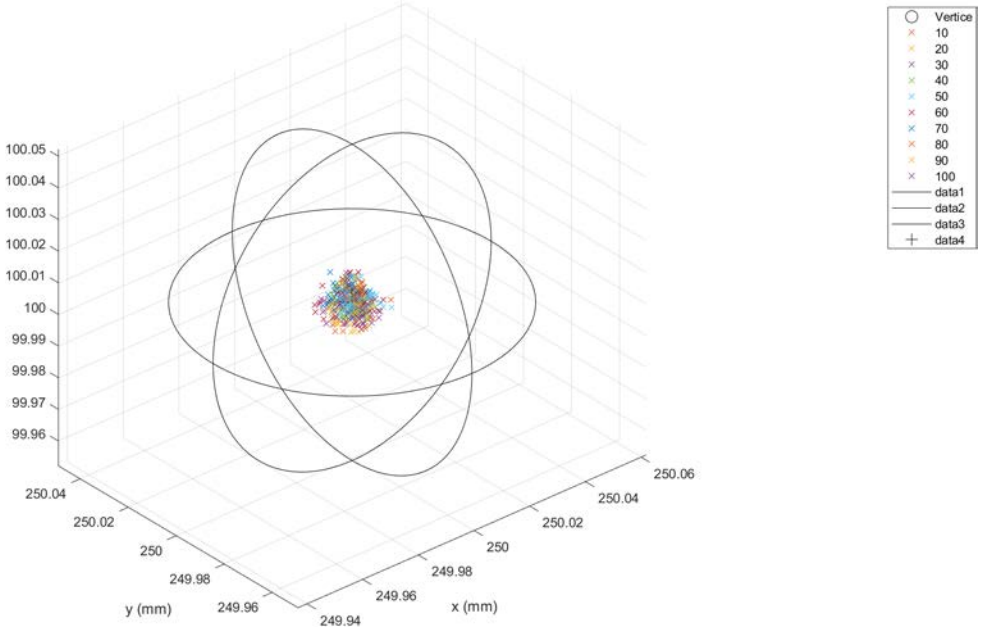


Figura 4.41: Vertice 250 250 con tutti i valori di Ta

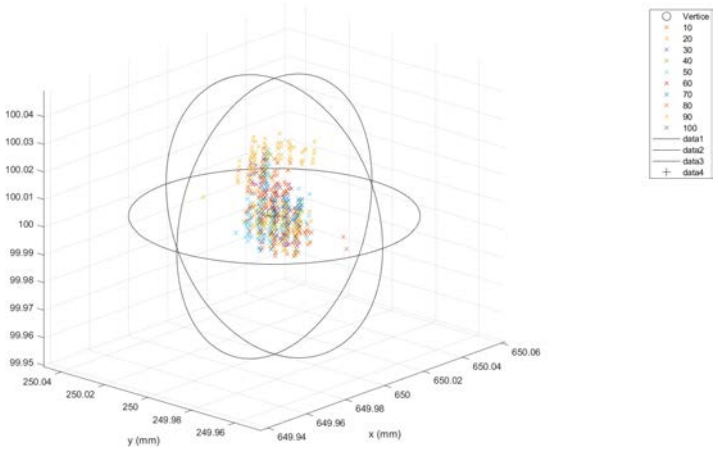


Figura 4.42: Vertice 650 250 con tutti i valori di Ta

Con il risultato delle prove si intuisce che non è la velocità che disturba direttamente la precisione e ripetibilità,ma fenomeni che andrebbero indagati più a fondo. Negli altri punti questo non si nota il che significa che per quei valori di parametri viene creata un disturbo particolarmente critico per il robot. Anche questo necessita un approfondimento. In ogni caso mediamente il valore di ripetibilità con probabilità del 99.8 % (quindi $3 * \sigma$) risulta sempre minore dei $\pm 0.05mm$ dichiarati dal costruttore. Un obbiettivo futuro è quello di andare a studiare ulteriore tipologie di movimento,come la funzione Line e Pline e vedere come queste si comportano al variare dei parametri. Anche in questo caso sarebbe bene provare percorsi diversi, andando anche far variare assieme il giunto 5 e 6, che durante queste prove, essendo sul piano, non venivano movimentati. Si riassumono in tabella i valori ricavati da tutte le prove per ogni vertice, senza quindi differenziare secondo i valori di velocità:

	Errore (mm)	Scarto quadratico medio (mm)
Vertice 1 (250,650)	0,0053	0,005
Vertice 2 (650,650)	0,023	0,01
Vertice 3 (650,250)	0,0033	0,0033
Vertice 4 (250,250)	0,0035	0,004

I valori sono compatibili con quelli dichiarati nel catalogo (ripetibilita = 0,05 mm) .

Capitolo 5

Telerobotica per TM5

Sono state usate le fonti [9] [7] [10] [11] [12]

In questo capitolo viene introdotto il concetto di tele-operatività, viene spiegato l'obbiettivo di comandare il robot e di come questo sia stato raggiunto.

5.1 Telerobotica

Con Telerobotica si intende la branca della robotica in cui il robot si occupa dell'implementazione meccanica del movimento, mentre l'operatore si occupa di qualsiasi pianificazione di programmazione o di decisione cognitiva. In poche parole l'uomo diventa la mente, mentre il robot è il corpo. La distanza è compensata dall'ambiente di telerobotica, descritto in figura.

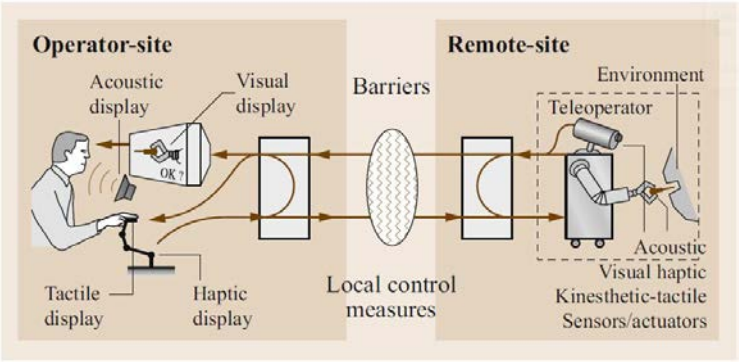


Figura 5.1: Struttura di un ambiente di telerobotica

Come si vedrà più avanti, nel nostro caso l'operator-site è composto da un joypad e lo script Matlab che visualizza il robot in tempo reale, mentre il Remote-site è la cella in cui è chiuso il robot.

5.2 Obiettivo

L'obbiettivo di questa tesi è quello di creare una piattaforma per tele-operazioni per TM5. In sintesi tramite un controller per xbox one si vogliono per compiere al robot i movimenti di traslazione e rotazione riferiti alla terna di base e alla terna tool, avendo poi anche la possibilità di muovere i singoli giunti.

Si è scelto un joypad per xbox one per la sua economicità e la semplicità di utilizzo.

Figura 5.2: Joypad Xbox One Front

Nel caso il Joypad Xbox non sia familiare la prima immagine rappresenta il fronte, mentre la seconda il retro del joypad.

Sono presenti due levette analogiche, distinte in destra e sinistra, sono presenti 4 tasti A,B,X,Y e un pad direzionale con 4 frecce. i due tasti centrali sono chiamati back (sinistra) e start (destra), nel retro si possono vedere le coppie di tasti Bumber e Trigger, distinte in Left e Right.

Figura 5.3: Joypad Xbox One Back

I movimenti vengono introdotti tramite Levette analogiche e trigger sul retro,perché permettono variazioni graduali dell'intensità della spinta, mentre gli altri sono pulsanti on/off che vengono usati per cambiare impostazioni di movimento.

Per la teleoperatività sono stati scritti 2 script diversi, il primo è un ambiente simulativo, il secondo è la vera piattaforma di teleoperazione.

5.3 Script DemoTM

Il primo script che viene descritto è DemoTM, questo è una simulazione dell'ambiente di lavoro del robot e lo script permette di muovere il modello tramite il Joypad.

Listing 5.1: Script DemoTM.m per simulazione ambiente

```
clc
clear
clear figure 1
%% Inizializzo figTM5a robot
hFig = figure(1);
hAx = gca;
%hFig.Position = [500 100 1000 850];          % x pc scuola
hFig.Position = [5 35 570 750];         % x pc mio
%hFig.Position = [-1280 -180 450 600]; % x schermo ausiliario
hFig.Color = 'w';             %coloro la figTM5a di bianco
hFig.NumberTitle = 'off';     %tolgo numero figTM5a
hFig.Name = 'Robot TM5';    %metto nome figTM5a

```

```
14 hold on
15 grid on
16 axis equal
17
18 xlabel('x [mm]')     % per dare nome agli assi
19 ylabel('y [mm]')
20 zlabel('z [mm]')
21 %% Definizione del robot
22
23 % inizializzo trasformata
24 hgTM5 = hgtransform();
25 fileDir = pwd;      % cartella corrente
26
27 robFile = 'TM5_900';  % nome del file del robot
28
29 datain.name = 'TM5_900';
30 %datain.toolFile = 'tool';
31 datain.hg = hgTM5;
32 datain.doframe = 1;
33
34 datain.hg = hgtransform('Matrix',eye(4),'Parent',hAx);
35 datain.h0 = hgtransform('Matrix',eye(4),'Parent',datain.hg);
36
37 % Creo l'oggetto
38 TM5 = KINpro(fileDir,robFile,datain);
39
40 %TM5.setTool(trans.Cardano(30,0,90+12,0,45,0)*trans.Z(180)) % tool
41 TM5.setWorld(trans.Cardano(0,0,0,0,0,0))        % terna base
42 TM5.setJ([45 0 90 0 90 0]);  % Impostazione giunti
43 % ———————— richiedere posizione e impostare setJ con quella—> poi
44 %  show
45 TM5.show();
46 dis.set3Dview(hAx,[60,10]) % angolo vista
47 xlim([-1000 1000])
48 ylim([-1000 1000])
49 zlim([0 1500])
50
51
52 dis.frame(eye(4),150,'o',TM5.hN);
53 dis.frame(eye(4),200,'o');
54
55 %% Controller
56 if exist('controller','var')
57     controller.stopConnection;
58 end
59 controller = xboxOneC();
60 global struttura
61 struttura.giuntoDaMuovere = 1;
```

```
struttura.frameWorld = 1;
struttura.framePezzo = 0;
struttura.SpaceOp = 1;
struttura.SpaceJ = 0;
struttura.rotZ=0;
struttura.traslZ=1;

if controller.isConnected
    disp('connesso!')
    controller.setBehaviour('trigger',{@movZ,TM5,controller})
    controller.setBehaviour('A',{@switchFrame,TM5,controller});
    controller.setBehaviour('B',{@switchZ,TM5,controller});
    controller.setBehaviour('Y',{@switchSpace,TM5,controller});
    controller.setBehaviour('leftT',{@traslXY,TM5,controller})
    controller.setBehaviour('rightT',{@rotXY,TM5,controller})
    controller.setBehaviour('Start',{@Homepos,TM5,controller})
    controller.setBehaviour('LeftBumper',{@MinusJoint,TM5,controller})
    controller.setBehaviour('RightBumper',{@PlusJoint,TM5,controller})
end


%% FUNZIONI PER IL MOVIMENTO

function Homepos (TM5,controller)
clear figure
TM5.setJ([45 0 90 0 90 0]);
TM5.update
TM5.show
end

function movZ(TM5,controller)
clear figure
global struttura

pZ = double(controller.Assi.trigger(1))/controller.minTrig*(abs(controller.Assi.trigger(1)) >
     controller.minTrig);
mZ = double(controller.Assi.trigger(2))/controller.minTrig*(abs(controller.Assi.trigger(2)) >
     controller.minTrig);
scalepZ = 0.5;
scalemZ = 0.5;

%TM5.q(struttura.giuntoDaMuovere) = TM5.q(struttura.giuntoDaMuovere) + (pZ*scalepZ - mZ*scalemZ);
%TM5.update
% TM5.setJ(q)
TM5.show;
if struttura.SpaceOp == 1
    if struttura.frameWorld == 1
      if struttura.rotZ == 1
```

```
        TM5.hN.Matrix = trans.Z(pZ*scalepZ-mZ*scalemZ)*TM5.hN.Matrix;
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
        TM5.setJ(q)
        TM5.update;
        TM5.show;
      else
        if struttura.traslZ == 1
        TM5.hN.Matrix = trans.P(0,0,pZ*scalepZ-mZ*scalemZ)*TM5.hN.Matrix;
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
        TM5.setJ(q)
        TM5.update;
        TM5.show;
        end
      end
   else
      if struttura.rotZ == 1
        TM5.hN.Matrix = TM5.hN.Matrix*trans.Z(pZ*scalepZ-mZ*scalemZ);
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
        TM5.setJ(q)
        TM5.update;
        TM5.show;
      else
        if struttura.traslZ == 1
        TM5.hN.Matrix = TM5.hN.Matrix*trans.P(0,0,pZ*scalepZ-mZ*scalemZ);
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
        TM5.setJ(q)
        TM5.update;
        TM5.show;
      end
      end
    end
else
    if struttura.SpaceJ == 1
        scalepZ = 0.2;
        scalemZ = 0.2;
TM5.q(struttura.giuntoDaMuovere) = TM5.q(struttura.giuntoDaMuovere) + (pZ*scalepZ - mZ*scalemZ);

TM5.update
TM5.setJ(q)
    end
end
end
function PlusJoint(TM5,controller)

global struttura
struttura.giuntoDaMuovere = struttura.giuntoDaMuovere + 1;
if struttura.giuntoDaMuovere>6
    struttura.giuntoDaMuovere = 1;
```

```
156 end
157 end
158 function MinusJoint(TM5,controller)
159 global struttura
160 struttura.giuntoDaMuovere = struttura.giuntoDaMuovere - 1;
161 if struttura.giuntoDaMuovere<1
162     struttura.giuntoDaMuovere = 6;
163 end
164 end
165
166 function switchSpace(TM5,controller)
167 global struttura
168 if struttura.SpaceOp == 1
169     struttura.SpaceJ = 1;
170     struttura.SpaceOp=0;
171 else
172     struttura.SpaceOp = 1;
173     struttura.SpaceJ = 1;
174 end
175
176 end
177
178 function switchFrame(TM5,controller)
179     global struttura
180 if struttura.frameWorld ==1
181     struttura.framePezzo = 1;
182     struttura.frameWorld = 0;
183 else
184     struttura.framePezzo = 0;
185     struttura.frameWorld = 1;
186 end
187
188 end
189 function switchZ(TM5,controller)
190     global struttura
191 if struttura.rotZ ==1
192     struttura.traslZ = 1;
193     struttura.rotZ = 0;
194 else
195     struttura.rotZ = 1;
196       struttura.traslZ = 0;
197 end
198 end
199 function traslXY(TM5,controller)
200 clear figure
201 global struttura
202 % ricava valore levetta
203 X = double(controller.Assi.leftT(1))/controller.minAxisL*(abs(controller.Assi.leftT(1)) > controller.
```

```
     minAxisL);
Y = double(controller.Assi.leftT(2))/controller.minAxisL*(abs(controller.Assi.leftT(2)) > controller.
     minAxisL);
scaleX = 0.9;
scaleY = 0.9;
%TM5.q(struttura.giuntoDaMuovere) = TM5.q(struttura.giuntoDaMuovere) + (pZ*scalepZ - mZ*scalemZ);
%TM5.update
if struttura.frameWorld ==1
    TM5.hN.Matrix = trans.P(X*scaleX,Y*scaleY,0)*TM5.hN.Matrix;
    q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
    TM5.setJ(q)
    TM5.update
    TM5.show
else
    TM5.hN.Matrix = TM5.hN.Matrix*trans.P(X*scaleX,Y*scaleY,0);
    q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
    TM5.setJ(q)
    TM5.update
    TM5.show
end
end
function rotXY(TM5,controller)
clear figure
global struttura
X = double(controller.Assi.rightT(1))/controller.minAxisR*(abs(controller.Assi.rightT(1)) >
     controller.minAxisR);
Y = double(controller.Assi.rightT(2))/controller.minAxisR*(abs(controller.Assi.rightT(2)) >
     controller.minAxisR);
scaleX = 0.2;
scaleY = 0.2;
if struttura.frameWorld ==1
    TM5.hN.Matrix = trans.X(X*scaleX)*trans.Y(Y*scaleY)*TM5.hN.Matrix;
    q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
    TM5.setJ(q)
    TM5.update ;
    TM5.show;
else if struttura.framePezzo ==1
    TM5.hN.Matrix = TM5.hN.Matrix*trans.X(X*scaleX)*trans.Y(Y*scaleY);
     q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
    TM5.setJ(q)
    TM5.update ;
    TM5.show;
    end
end
end
```

Le prime 20 righe riguardano la definizione della parte grafica, successivamente si inizializzano i parametri necessari alla classe KinPRO per creare l'oggetto

matlab TM5 con le caratteristiche del TM5.

Una volta creato il robot si crea l'oggetto controller, la funzione if alla riga 52 serve per scollegare e coi collegare il controller per evitare errori.

Viene definita dunque la Variabile Globale *struttura*, che contiene le informazioni sul tipo di movimento che un azionamento del joypad esegue.

Una variabile globale può essere usata e modificata all'interno delle funzioni senza che essa sia una variabile di ingresso. Deve tuttavia essere sempre richiamata all'inizio di ogni funzione in cui viene utilizzata.

I vari argomenti che sono salvati in *struttura* sono:

- $struttura.giuntoDaMuovere = 1$ è il giunto selezionato quando si vuole far muovere i giunti singolarmente, inizializzato ad 1
- $struttura.frameWorld = 1$ se il tipo di movimento è cartesiano e la terna di riferimento è la terna World
- $struttura.framePezzo = 0$ se il tipo di movimento è cartesiano e la terna di riferimento è la terna Tool, inizializzato a zero perché $struttura.frameWorld = 1$
- $struttura.SpaceOp = 1$ se si effettuano movimenti cartesiani, inizializzato ad 1
- $struttura.SpaceJ = 0$ se si effettuano movimenti dei singoli giunti, inizializzato a zero perché $struttura.SpaceOp = 1$
- $struttura.rotZ = 0$ se l'azionamento dei due trigger deve far ruotate attorno all'asse Z della terna selezionata, inizializzato a 0
- $struttura.rotZ = 0$ se l'azionamento dei due trigger deve far compiere una traslazione in direzione Z della terna selezionata,inizializzato ad 1

Nella funzione if successiva, attiva se presente un controller, si associano ai tasti del joypad le funzioni di movimento che sotto sono esplicitate.

Homepos

La funzione Homepos è associata al tasto start, questa sposta il robot velocemente nella posizione nello spazio dei giunti che corrisponde ad una posizione familiare dalla quale è poi possibile muoversi facilmente, non rappresenta la posizione Home del robot.

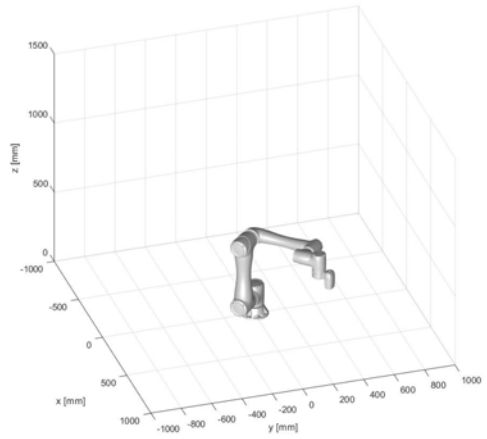


Figura 5.4: Posizione Homepos

movZ

La funzione movZ è associata ai due trigger, destro e sinistro. Questa viene usata per traslare e ruotare in riferimento all'asse Z della terna prefissata e muovere i giunti in caso di movimento dei singoli giunti.

Come primo caso si ricavano dall'oggetto controller i valori pZ e mZ, ovvero quanto a fondo i trigger sono premuti. Questi valori vengono poi scalati tramite moltiplicazione con le variabili scalepZ e scalemZ entrambi inizializzati con il valore di 0.5. Tramite if alla riga 101 si valuta se si sta operando nello spazio operativo oppure in quello dei giunti e nel primo caso si valuta la terna di riferimento, se il movimento sarà di rotazione oppure no, sono dunque 3 if in cascata.

Per una rotazione attorno all'asse si sostituisce la matrice della terna tool TM5.hN.Matrix con il risultato di sé stessa premoltiplicata con una matrice di rotazione attorno all'asse Z. L'angolo di rotazione è dato da $(pZ * scalepZ - mZ *$

$scalemZ$) in modo che i due trigger siano i due versi di rotazione dell'angolo. Tramite cinematica inversa si calcolano gli angoli (per la configurazione si usa la funzione getconf(obj) della classe KinPRO), usando setJ, update e show si impostano gli angoli trovati all'oggetto TM5 e l'immagine si aggiorna con la nuova posizione.

Nel caso di una traslazione l'operazione è la stessa, ma con trans.P di $(0, 0, pZ * scalepZ - mZ * scalemZ)$ anziché il precedente trans.Z.

Nel caso in cui la terna di riferimento sia la terna tool il procedimento è uguale, ma l'ordine delle matrici nella moltiplicazione è invertito.

Passando invece alla rotazione del singolo giunto è sufficiente impostare il giunto con numero $struttura.giuntoDaMuovere$ con il valore attuale sommato a $(pZ * scalepZ - mZ * scalemZ)$, similmente a quanto fatto per la traslazione.

PlusJoint

La funzione PlusJoint è associata al tasto right bumper, si utilizza per aumentare di 1 il valore del giunto da muovere.

Nel caso sia uguale a 6, si ritorna al valore 1.

MinusJoint

La funzione MinusJoint è associata al tasto left bumper, si utilizza per diminuire di 1 il valore del giunto da muovere.

Nel caso sia uguale a 1, si ritorna al valore 6.

switchSpace

La funzione switchSpace è associata al tasto Y, si utilizza per cambiare lo spazio di movimento, se lo spazio dei giunti oppure lo spazio operativo.

switchFrame

La funzione switchFrame è associata al tasto A, si utilizza per cambiare la terna di riferimento, se la terna world oppure la terna tool.

switchZ

La funzione switchZ è associata al tasto B,cambia il tipo di movimento dell'asse Z da traslazione a rotazione e viceversa.

traslXY

La funzione traslXY è associata alla levetta analogica sinistra e serve a muovere il tool in direzione X e Y rispetto alla terna selezionata.

Dopo aver dichiarato la variabile struttura, si richiamano i valori X e Y dall'oggetto controller, simili ai mZ e pZ visti precedentemente.

Questi verranno poi scalati rispetto a scaleX e scaleY.

Nel caso di terna di riferimento world si premoltiplica la matrice del tool TM5.hN.Matrix con una matrice di traslazione di coordinate $(scaleX*X, scaleY*Y, 0)$. Tramite cinematica inversa si calcolano gli angoli (per la configurazione si usa la funzione getconf(obj) della classe KinPRO), usando setJ, update e show si impostano gli angoli trovati all'oggetto TM5 e l'immagine si aggiorna con la nuova posizione Nel caso in cui la terna di riferimento sia la terna tool il procedimento è uguale, ma l'ordine delle matrici nella moltiplicazione è invertito.

rotXY

La funzione rotXY è associata alla levetta analogica destra e serve a ruotare il tool del robot in direzione X e Y rispetto alla terna selezionata.

Dopo aver dichiarato la variabile struttura, si richiamano i valori X e Y dall'oggetto controller, simili ai mZ e pZ visti precedentemente.

Questi verranno poi scalati rispetto a scaleX e scaleY.

Nel caso di terna di riferimento world si premoltiplica la matrice TM5.hN.Matrix con una matrice di rotazione su X di $X*scaleX$ e poi su Y di $Y*scaleY$. Tramite cinematica inversa si calcolano gli angoli (per la configurazione si usa la funzione getconf(obj) della classe KinPRO), usando setJ, update e show si impostano gli angoli trovati all'oggetto TM5 e l'immagine si aggiorna con la nuova posizione. Nel caso in cui la terna di riferimento sia la terna tool il procedimento è uguale, ma l'ordine delle matrici nella moltiplicazione è invertito.

5.4 Script TeleJoypadMatrici

Il secondo script è TeleJoypadMatrici.m ed è la vera piattaforma di telerobotica finale. Questa permette di muovere il robot tramite joypad e di avere nello schermo la simulazione in tempo reale del robot.

Listing 5.2: Script TeleJoypadMatrici.m per telerobotica

```
1
2  clc
3
4  clear figure 1
5  %% Inizializzo figTM5a robot
6  hFig = figure(1);
7  hAx = gca;
8  global struttura
9  %hFig.Position = [500 100 1000 850];          % x pc scuola
10 hFig.Position = [5 35 570 750];         % x pc mio
11 %hFig.Position = [-1280 -180 450 600]; % x schermo ausiliario
12 hFig.Color = 'w';             %coloro la figTM5a di bianco
13 hFig.NumberTitle = 'off';     %tolgo numero figTM5a
14 hFig.Name = 'Robot TM5';    %metto nome figTM5a
15
16 hold on
17 grid on
18 axis equal
19
20 xlabel('x [mm]')     % per dare nome agli assi
21 ylabel('y [mm]')
22 zlabel('z [mm]')
23
24
25 struttura.hRosso.XData=[];  % prendo la linea di colore rosso
26 struttura.hRosso.YData=[];
27 struttura.hRosso.ZData=[];
28
29 struttura.hBlu.XData = [];  % prendo la linea di colore blu
30 struttura.hBlu.YData = [];
31 struttura.hBlu.ZData = [];
32
33 struttura.qIdeal=[];
34 struttura.qReal=[];
35
36
37 %% Definizione del robot
38
39 % inizializzo trasformata
40 hgTM5 = hgtransform();
```

```
fileDir = pwd;      % cartella corrente

robFile = 'TM5_900';  % nome del file del robot

datain.name = 'TM5_900';
%datain.toolFile = 'tool';
datain.hg = hgTM5;
datain.doframe = 1;

datain.hg = hgtransform('Matrix',eye(4),'Parent',hAx);
datain.h0 = hgtransform('Matrix',eye(4),'Parent',datain.hg);

% Creo l'oggetto
TM5 = KINpro(fileDir,robFile,datain);

%TM5.setTool(trans.Cardano(30,0,90+12,0,45,0)*trans.Z(180)) % tool
TM5.setWorld(trans.Cardano(0,0,0,0,0,0))        % terna base
TM5.setJ(TM.ask(connessione,1)); % Impostazione giunti

% ---------------- richiedere posizione e impostare setJ con quella--> poi
%  show
TM5.show();
dis.set3Dview(hAx,[60,10]) % angolo vista
xlim([-1000 1000])
ylim([-1000 1000])
zlim([0 1500])

dis.frame(eye(4),150,'o',TM5.hN);
dis.frame(eye(4),200,'o');



%% Controller
if exist('controller','var')
    controller.stopConnection;
end
controller = xboxOneC();

struttura.giuntoDaMuovere = 1;
struttura.frameWorld = 1;
struttura.framePezzo = 0;
struttura.SpaceOp = 1;
struttura.SpaceJ = 0;
struttura.rotZ=0;
struttura.traslZ=1;


%% Parametri di moviento
```

```
89
90  struttura.Ta=100;
91  struttura.racc=100;
92  struttura.vel=100;
93  struttura.PrecPos=false;
94  struttura.velj=[ 10 10 10 20 20 20 ];
95  struttura.Forza=[]
96
97
98  %% Collegamento controller
99  if controller.isConnected
100     disp('connesso!')
101     controller.setBehaviour('trigger',{@movZ,TM5,controller,connessione})
102     controller.setBehaviour('A',{@switchFrame,TM5,controller,connessione});
103     controller.setBehaviour('B',{@switchZ,TM5,controller,connessione});
104     controller.setBehaviour('Y',{@switchSpace,TM5,controller,connessione});
105     controller.setBehaviour('leftT',{@traslXY,TM5,controller,connessione})
106     controller.setBehaviour('rightT',{@rotXY,TM5,controller,connessione})
107     controller.setBehaviour('Start',{@Homepos,TM5,controller,connessione})
108     controller.setBehaviour('LeftBumper',{@MinusJoint,TM5,controller,connessione})
109     controller.setBehaviour('RightBumper',{@PlusJoint,TM5,controller,connessione})
110
111 end
112
113
114 %% FUNZIONI PER IL MOVIMENTO
115
116 function Homepos (TM5,controller,connessione)
117 clear figure
118 global struttura
119 q=[45 0 90 0 90 0];
120 TM.PTP(connessione,'JPP',q,struttura.vel/2,struttura.Ta,struttura.racc,struttura.PrecPos)
121
122         TM5.setJ(q)
123         TM5.update;
124         TM5.show;
125
126
127 end
128
129 function movZ(TM5,controller,connessione)
130 clear figure
131 global struttura
132
133 pZ = double(controller.Assi.trigger(1))/controller.minTrig*(abs(controller.Assi.trigger(1)) >
      controller.minTrig);
134 mZ = double(controller.Assi.trigger(2))/controller.minTrig*(abs(controller.Assi.trigger(2)) >
      controller.minTrig);
```

```
scalepZ =2;
scalemZ =2;
pause(0.05)
%TM5.q(struttura.giuntoDaMuovere) = TM5.q(struttura.giuntoDaMuovere) + (pZ*scalepZ - mZ*scalemZ);
%TM5.update
% TM5.setJ(q)
TM5.show;
if struttura.SpaceOp == 1
    if struttura.frameWorld == 1
      if struttura.rotZ == 1

        xyz=TM5.matrici.TU0(1:3,4) ;
        TM5.hN.Matrix=(trans.Z(pZ*scalepZ-mZ*scalemZ))*TM5.hN.Matrix;
        %trans.P(xyz(1),xyz(2),xyz(3))*
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,TM5.conf,TM5.q,TM5.dati);
        TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)

        TM5.setJ(q)
        TM5.update;
        TM5.show;


      else
        if struttura.traslZ == 1
        TM5.hN.Matrix = trans.P(0,0,pZ*scalepZ-mZ*scalemZ)*TM5.hN.Matrix;
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
        TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)
        TM5.setJ(q)
        TM5.update;
        TM5.show;
        end
      end
   else
      if struttura.rotZ == 1
        TM5.hN.Matrix = TM5.hN.Matrix*trans.Z(pZ*scalepZ-mZ*scalemZ);
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,TM5.conf,TM5.q,TM5.dati);
        TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)

        TM5.setJ(q)
        TM5.update;
        TM5.show;

      else
        if struttura.traslZ == 1
        TM5.hN.Matrix = TM5.hN.Matrix*trans.P(0,0,pZ*scalepZ-mZ*scalemZ);
        q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
               TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos
                    )
```

```
182
183         TM5.setJ(q)
184         TM5.update;
185         TM5.show;
186
187
188       end
189       end
190     end
191 else
192     if struttura.SpaceJ == 1
193
194     scalepZ = 0.2;
195         scalemZ = 0.2;
196 TM5.q(struttura.giuntoDaMuovere) = TM5.q(struttura.giuntoDaMuovere) + (pZ*scalepZ - mZ*scalemZ);
197 TM5.update
198         %TM.Move_PTP(connessione,'JPP',struttura.cooj,struttura.velj(struttura.giuntoDaMuovere),
              struttura.Ta,struttura.racc,struttura.PrecPos);%rotazione assoluta attorno a Z
199
200         TM.PTP(connessione,'JPP',TM5.q,struttura.vel/5,struttura.Ta,struttura.racc,struttura.PrecPos)
201
202         TM5.setJ(q)
203         TM5.update;
204         TM5.show;
205
206
207     end
208 end
209 end
210 function PlusJoint(TM5,controller,connessione)
211
212 global struttura
213 struttura.giuntoDaMuovere = struttura.giuntoDaMuovere + 1;
214 if struttura.giuntoDaMuovere>6
215     struttura.giuntoDaMuovere = 1;
216 end
217 end
218 function MinusJoint(TM5,controller,connessione)
219 global struttura
220 struttura.giuntoDaMuovere = struttura.giuntoDaMuovere - 1;
221 if struttura.giuntoDaMuovere<1
222     struttura.giuntoDaMuovere = 6;
223 end
224 end
225
226 function switchSpace(TM5,controller,connessione)
227 global struttura
228 if struttura.SpaceOp == 1
```

```
229     struttura.SpaceJ = 1;
230     struttura.SpaceOp=0;
231 else
232     struttura.SpaceOp = 1;
233     struttura.SpaceJ = 1;
234 end
235 
236 end
237 
238 function switchFrame(TM5,controller,connessione)
239     global struttura
240 if struttura.frameWorld ==1
241     struttura.framePezzo = 1;
242     struttura.frameWorld = 0;
243 else
244     struttura.framePezzo = 0;
245     struttura.frameWorld = 1;
246 end
247 
248 end
249 function switchZ(TM5,controller,connessione)
250     global struttura
251 if struttura.rotZ ==1
252     struttura.traslZ = 1;
253     struttura.rotZ = 0;
254 else
255     struttura.rotZ = 1;
256       struttura.traslZ = 0;
257 end
258 end
259 function traslXY(TM5,controller,connessione)
260 pause(0.05)
261 clear figure
262 global struttura
263 % ricava valore levetta
264 X = double(controller.Assi.leftT(1))/controller.minAxisL*(abs(controller.Assi.leftT(1)) > controller.
      minAxisL);
265 Y = double(controller.Assi.leftT(2))/controller.minAxisL*(abs(controller.Assi.leftT(2)) > controller.
      minAxisL);
266 scaleX = 2;
267 scaleY = 2;
268 %TM5.q(struttura.giuntoDaMuovere) = TM5.q(struttura.giuntoDaMuovere) + (pZ*scalepZ - mZ*scalemZ);
269 %TM5.update
270 if struttura.frameWorld ==1
271     TM5.hN.Matrix = trans.P(X*scaleX,Y*scaleY,0)*TM5.hN.Matrix;
272     q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
273         TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)
274 
```

```
275         TM5.setJ(q)
276         TM5.update;
277         TM5.show;
278 
279 else
280     TM5.hN.Matrix = TM5.hN.Matrix*trans.P(X*scaleX,Y*scaleY,0);
281     q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
282           TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)
283 
284         TM5.setJ(q)
285         TM5.update;
286         TM5.show;
287 
288 end
289 end
290 function rotXY(TM5,controller,connessione)
291 pause(0.045)
292 clear figure
293 global struttura
294 X = double(controller.Assi.rightT(1))/controller.minAxisR*(abs(controller.Assi.rightT(1)) >
     controller.minAxisR);
295 Y = double(controller.Assi.rightT(2))/controller.minAxisR*(abs(controller.Assi.rightT(2)) >
     controller.minAxisR);
296 scaleX = 0.3;
297 scaleY = 0.3;
298 if struttura.frameWorld ==1
299     TM5.hN.Matrix = trans.X(X*scaleX)*trans.Y(Y*scaleY)*TM5.hN.Matrix;
300     q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
301            TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)
302 
303         TM5.setJ(q)
304         TM5.update;
305         TM5.show;
306 
307 
308 else if struttura.framePezzo ==1
309     TM5.hN.Matrix = TM5.hN.Matrix*trans.X(X*scaleX)*trans.Y(Y*scaleY);
310      q=TM5.invKinPROurTM5(TM5.hN.Matrix,getConf(TM5),TM5.q,TM5.dati);
311           TM.PTP(connessione,'JPP',q,struttura.vel,struttura.Ta,struttura.racc,struttura.PrecPos)
312 
313         TM5.setJ(q)
314         TM5.update;
315         TM5.show;
316 
317 
318 
319 
320     end
```

```
321 end
322 end
```

La parte prima delle descrizioni delle funzioni è simile allo script precedente, ma con alcune differenze che è necessario sottolineare.

Alle righe 25-35 vengono inizializzate alcuni campi in *Struttura* per memorizzare la traiettoria del modello e quella del robot reale, comprese le coordinate espresse nel mondo dei giunti. Alla riga 58 anziché impostare il modello a angoli prefissati, viene fatta richiesta al robot della posizione dei giunti e tramite funzione setJ si da all'oggetto TM5 la stessa posizione del robot reale, poi tramite il comando dis.frame si visualizza la terna World e la terna tool.

A questo punto si definiscono gli altri campi della variabile *struttura* dove, prima degli elementi già visti nel precedente script, sono presenti i parametri di movimento:

- $struttura.Ta = 250$ è il valore del tempo di accelerazione dei movimenti in ms
- $struttura.racc = 100$ è il valore in percentuale del raccordo tra le varie traiettorie
- $struttura.vel = 300$ valore di velocità in mm/s
- $struttura.PrecPos = false$abilita la modalità "precise positioning", true invece la disabilita
- $struttura.velJ = [101010202020]$ un vettore contenente le velocità da usare per il movimento dei singoli giunti, nel caso si voglia muovere un giunto alla volta
- $struttura.Forza = []$ Un vettore dove nel caso è possibile salvare i valori di forza.
- $struttura.giuntoDaMuovere = 1$ è il giunto che si muove quando si vogliono far muovere i giunti singolarmente, inizializzato ad 1
- $struttura.frameWorld = 1$ se il tipo di movimento è cartesiano e la terna di riferimento è la terna World

- $struttura.framePezzo = 0$ se il tipo di movimento è cartesiano e la terna di riferimento è la terna Tool, inizializzato a zero perché $struttura.frameWorld = 1$

- $struttura.SpaceOp = 1$ se si effettuano movimenti cartesiani, inizializzato ad 1

- $struttura.SpaceJ = 0$ se si effettuano movimenti dei singoli giunti

- $struttura.rotZ = 0$ se l'azionamento dei due trigger deve far ruotate attorno all'asse Z della terna selezionata, inizializzato a 0

- $struttura.rotZ = 0$ se l'azionamento dei due trigger deve far compiere una traslazione in direzione Z della terna selezionata,inizializzato ad 1

Nella funzione if successiva, attiva se presente un controller, si associano ai tasti del joypad le funzioni di movimento che sotto sono esplicitate.

Homepos

La funzione Homepos è associata al tasto start, questa sposta il robot tramite comando PTP nella posizione nello spazio [450900900], non rappresenta la posizione Home del robot.

Contemporaneamente, il modello del robot viene impostato nella stessa posizione.

movZ

La funzione movZ è associata ai due trigger, destro e sinistro. Questa viene usata per traslare e ruotare in riferimento all'asse Z della terna prefissata e muovere i giunti in caso di movimento dei singoli giunti. Mentre i trigger sono premuti viene sempre richiesta la posizione in modo che venga rappresentata fedelmente in figura 1.

Nel caso di movimento cartesiano, si ricavano dall'oggetto controller i valori pZ e mZ, ovvero quanto affondo i trigger sono premuti. Questi valori vengono poi scalati tramite moltiplicazione con le variabili scalepZ e scalemZ, entrambi

inizializzati con il valore di 0.5. Tramite if alla riga 114 si valuta se si sta operando nello spazio operativo oppure in quello dei giunti e nel primo caso si valuta la terna di riferimento e se il movimento sarà di rotazione oppure no, sono dunque 3 if in cascata.

Per una rotazione attorno all'asse cartesiano Z si procede come nello script precedente: si sostituisce la matrice della terna tool TM5.hN.Matrix con il risultato di sé stessa premoltiplicata con una matrice di rotazione attorno all'asse Z. L'angolo di rotazione è dato da $(pZ * scalepZ - mZ * scalemZ)$ in modo che i due trigger siano i due versi di rotazione dell'angolo. Tramite cinematica inversa si calcolano gli angoli (per la configurazione si usa la funzione getconf(obj) della classe KinPRO), usando setJ, update e show si impostano gli angoli trovati all'oggetto TM5 e l'immagine si aggiorna con la nuova posizione.

In aggiunta si esegue un comando PTP in direzione delle coordinate q appena ricavate dalla cinematica indiretta, con i parametri di movimento già definiti, in modo che il robot segua il modello ideale.

Nel caso di una traslazione l'operazione è la stessa, ma con trans.P di $(0, 0, pZ * scalepZ - mZ * scalemZ)$ anziché il precedente trans.Z. Sempre si calcolano poi le coordinate dei giunti in q, e, dopo aver aggiornato il modello, si muove il robot tramite PTP.

Per muovere singolo giunto si cambia il singolo giunto del modello e poi si inviano le coordinate TM5.q tramite PTP, dopo di che si aggiorna il modello. Si chiede dunque la posizione del robot e si aggiorna il modello.

PlusJoint

La funzione PlusJoint è associata al tasto right bumper, si utilizza per aumentare di 1 il valore del giunto da muovere.

Nel caso sia uguale a 6, si ritorna al valore 1.

MinusJoint

La funzione MinusJoint è associata al tasto left bumper, si utilizza per diminuire di 1 il valore del giunto da muovere.

Nel caso sia uguale a 1, si ritorna al valore 6.

switchSpace

La funzione switchSpace è associata al tasto Y, si utilizza per cambiare lo spazio di movimento, se lo spazio dei giunti oppure lo spazio operativo.

switchFrame

La funzione switchFrame è associata al tasto A, si utilizza per cambiare la terna di riferimento, se la terna world oppure la terna tool.

switchZ

La funzione switchZ è associata al tasto B,cambia il tipo di movimento dell'asse Z da traslazione a rotazione e viceversa.

traslXY

La funzione traslXY è associata alla levetta analogica sinistra e serve a muovere il tool in direzione X e Y rispetto alla terna selezionata.

Dopo aver dichiarato la variabile struttura, si richiamano i valori X e Y dall'oggetto controller, simili ai mZ e pZ visti precedentemente.

Questi verranno poi scalati rispetto a scaleX e scaleY.

Nel caso di traslazione in riferimento dalla terna world si premoltiplica la matrice del tool TM5.hN.Matrix con una matrice di traslazione di coordinate $(scaleX * X, scaleY * Y, 0)$. Tramite cinematica inversa si calcolano gli angoli (per la configurazione si usa la funzione getconf(obj) della classe KinPRO), usando setJ, update e show si impostano gli angoli trovati all'oggetto TM5 e l'immagine si aggiorna con la nuova posizione. Tramite movimento PTP, con le coordinate appena trovate, si muove il robot reale in modo che segua il modello. Nel caso in cui la terna di riferimento sia la terna tool il procedimento è uguale, ma l'ordine delle matrici nella moltiplicazione è invertito. Poi sempre per PTP si muove il TM vero.

rotXY

La funzione rotXY è associata alla levetta analogica destra e serve a ruotare il tool del robot in direzione X e Y rispetto alla terna selezionata.

Dopo aver dichiarato la variabile struttura, si richiamano i valori X e Y dall'oggetto controller, simili ai mZ e pZ visti precedentemente.

Questi verranno poi scalati rispetto a scaleX e scaleY.

Nel caso di terna di riferimento world si premoltiplica la matrice TM5.hN.Matrix con una matrice di rotazione su X di $X * scaleX$ e poi su Y di $Y * scaleY$. Tramite cinematica inversa si calcolano gli angoli (per la configurazione si usa la funzione getconf(obj) della classe KinPRO), usando setJ, update e show si impostano gli angoli trovati all'oggetto TM5 e l'immagine si aggiorna con la nuova posizione. Tramite movimento PTP, con le coordinate appena trovate, si muove il robot reale in modo che segua il modello.

Nel caso in cui la terna di riferimento sia la terna tool il procedimento è uguale, ma l'ordine delle matrici nella moltiplicazione è invertito. Poi sempre per PTP si muove il TM vero.

5.4.1 Paragone Modello e Robot

In questa sezione si esegue un paragone tra il modello ideale e il Robot reale, andando a compiere un percorso casuale con tramite Joypad; registrando posizione nel mondo dei giunti e cartesiana, sia del modello sia del robot: il modello sarà rappresentato in rosso, mentre il Robot reale in Blu.

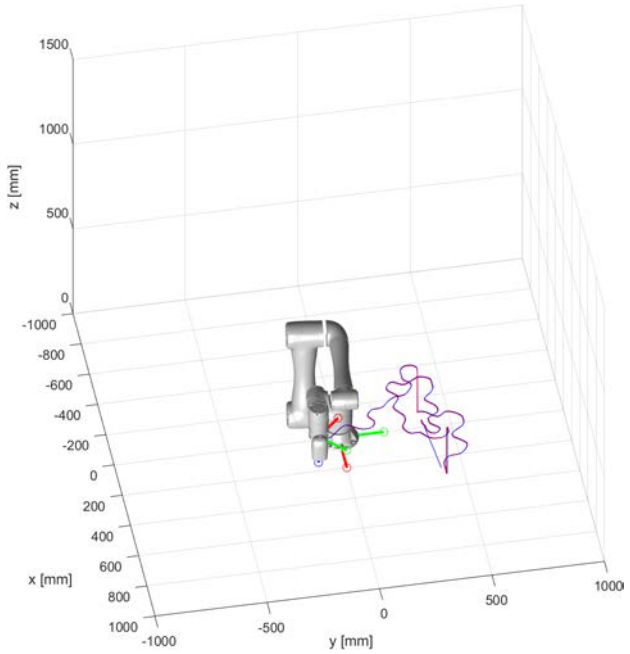


Figura 5.5: Prima vista del percorso causale

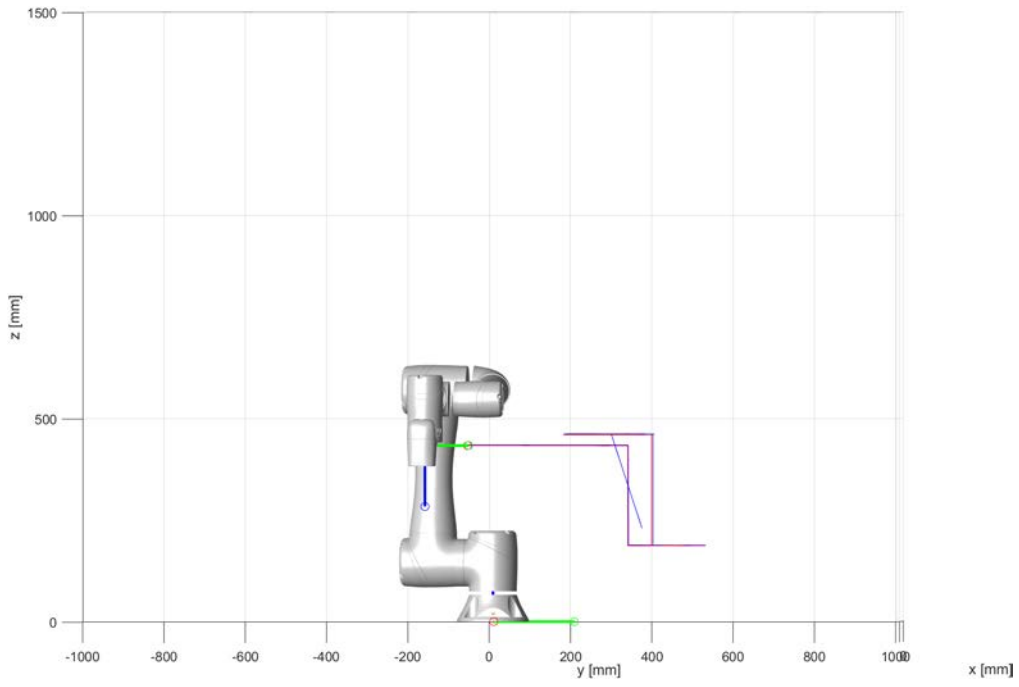
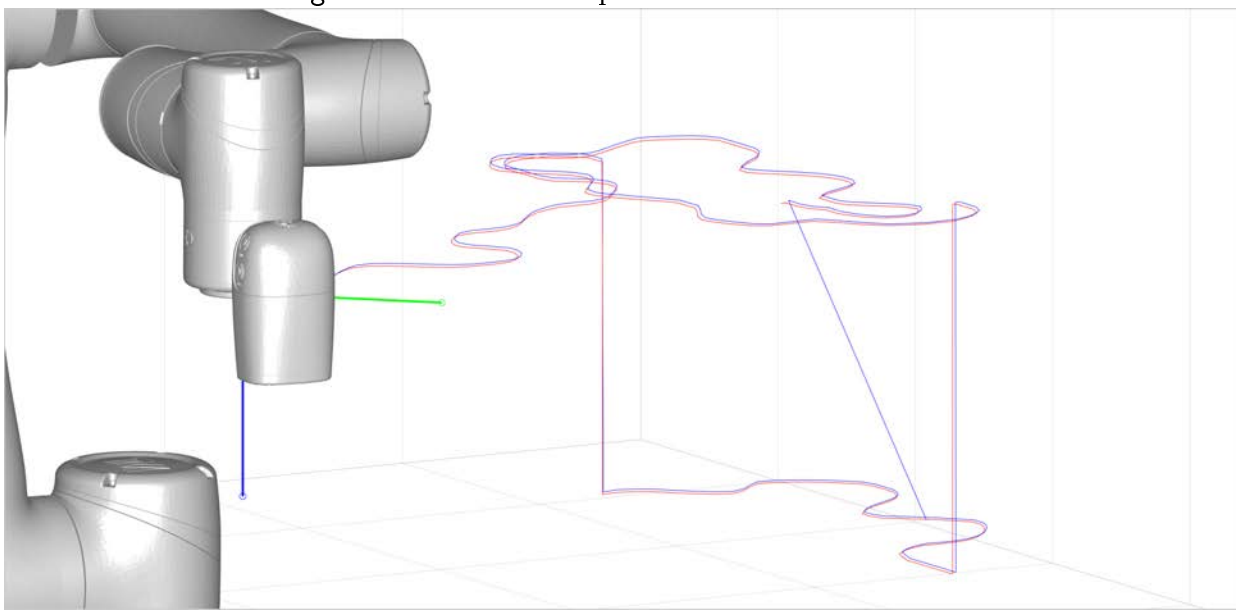


Figura 5.6: Seconda vista del percorso casuale

Figura 5.7: Terza vista percorso casuale



Nella terza vista più delle altre è possibile evidenziare la leggera differenza tra i due percorsi, questo è causato da un naturale ritardo di comunicazione tra la piattaforma e il robot.

Per andare a quantificare questo ritardo si prendono in esame i profili delle posizioni dei singoli giunti e controlla di quanto siano sfasati i due profili.

Figura 5.8: Profilo posizione giunto 1

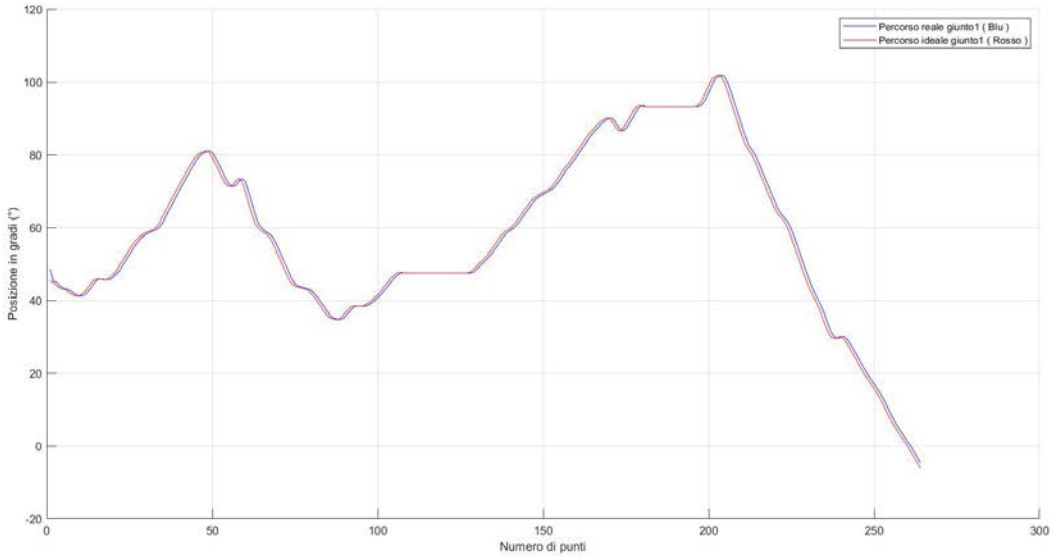


Figura 5.9: Profilo posizione giunto 2

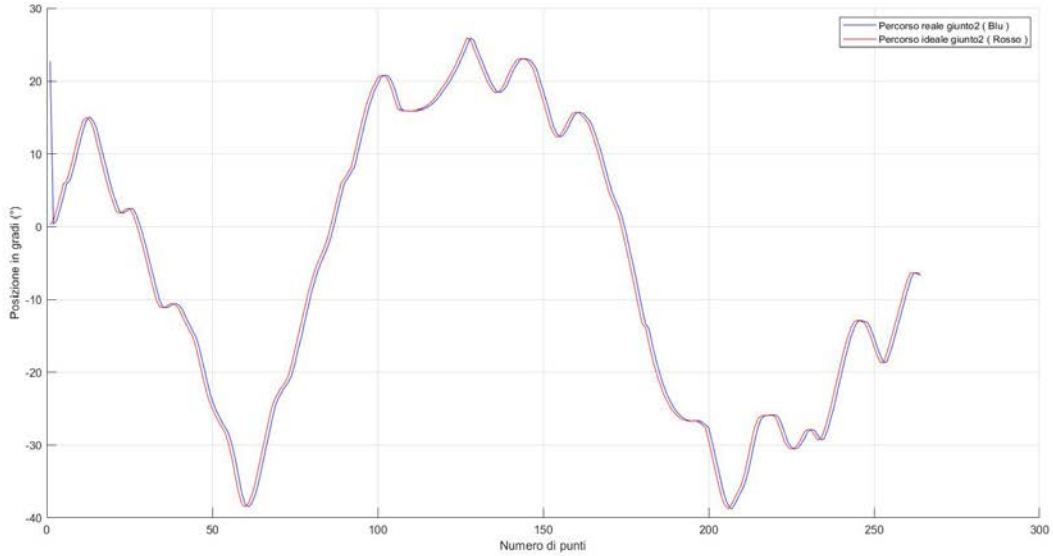


Figura 5.10: Profilo posizione giunto 3

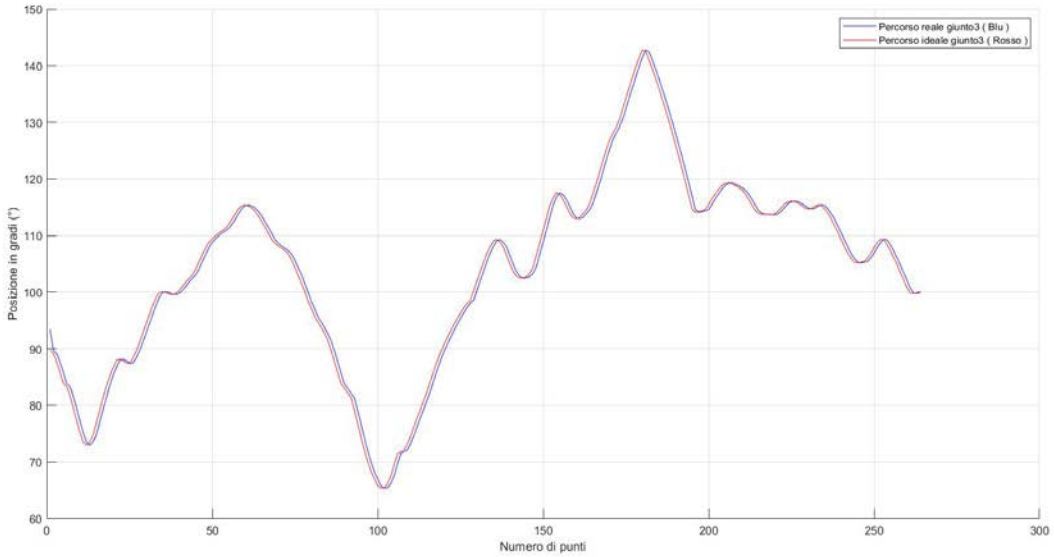


Figura 5.11: Profilo posizione giunto 4

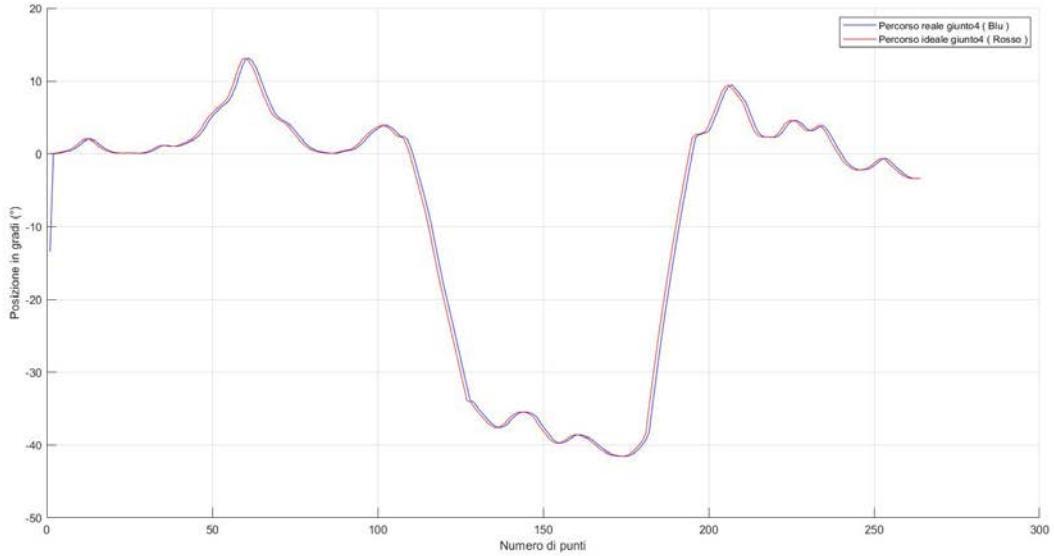


Figura 5.12: Profilo posizione giunto 5

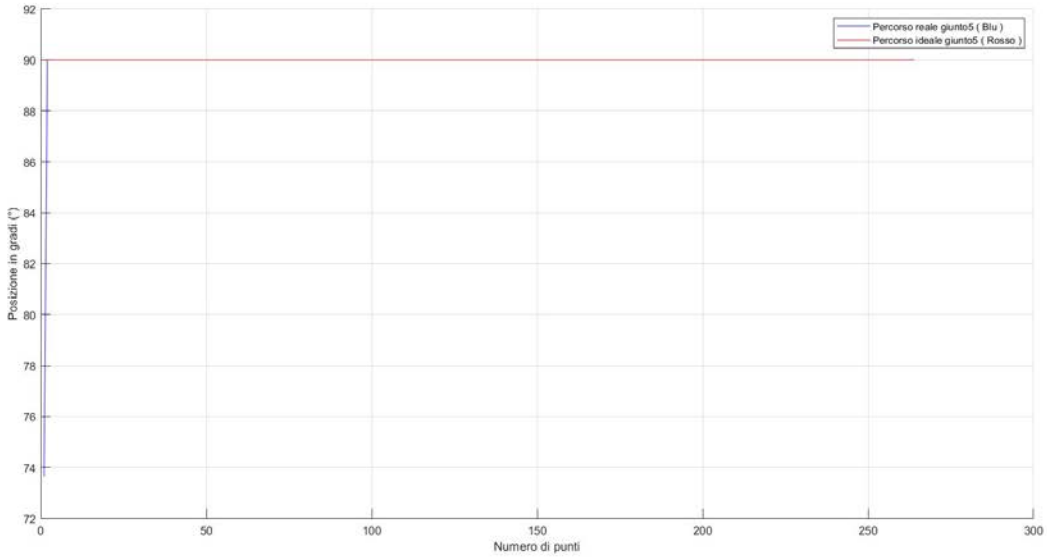
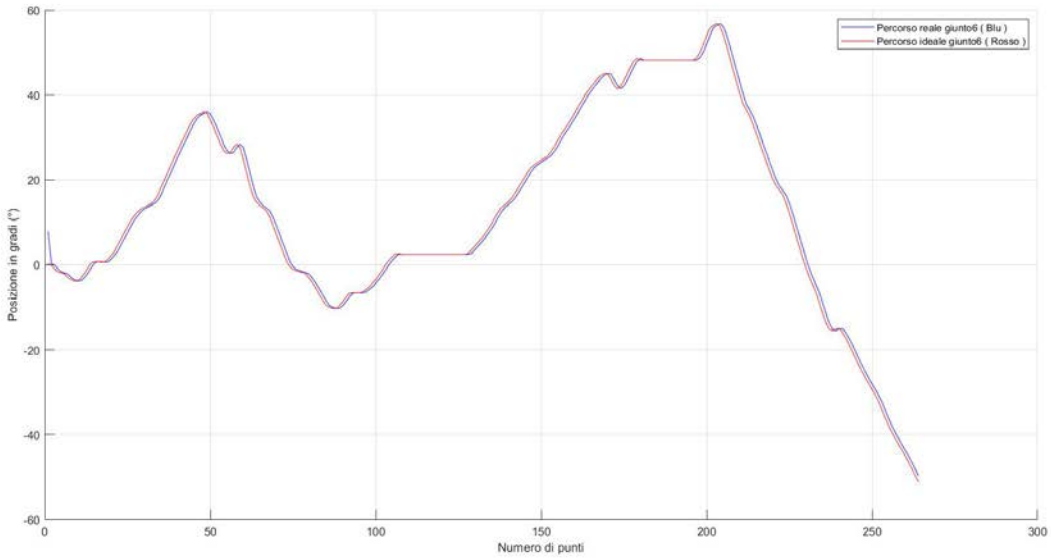


Figura 5.13: Profilo posizione giunto 6



Le due curve sono circa sfasate di circa un punto, la frequenza di trasmissione non è un valore pre-impostato, ma dipende da matlab e da quanto velocemente riesce a ripetere la funzione associata al pulsante premuto. Nonostante ciò è possibile leggendo il log di sistema del Robot andare a contare quante volte mediamente al robot viene richiesta la sua posizione e, per quanto è stato osservato, questa viene richiesta e spedita circa 3-4 volte al secondo, dunque si presuppone un ritardo si circa 0,3 secondi.

Conclusioni

L'obbiettivo del creare una piattaforma per tele-operazioni è stato raggiunto, ma le possibilità di lavoro e di miglioramento sono ancora molte.

Il lag di movimento non è trascurabile per applicazioni di precisione, anche se per movimenti semplici, dunque di ordine di grandezza sopra il centimetro, non si hanno problemi. Ogni movimento è formato da diversi tratti che vengono poi raccordati tra di loro. Ad ogni intervallo di comunicazione questo tratto deve essere concluso, in modo tale che alla comunicazione successiva viene ricevuto il nuovo tratto da percorre e i due vengano raccordati fluentemente. Se volessi aumentare la frequenza dovrei quindi ridurre la lunghezza dei tratti percorsi, che nel codice vengono rappresentati dal fattore di scala. Facendo così però si sfora nel campo in cui il profilo di velocità del singolo tratto diventa triangolare e il Ta non è sufficiente a raggiungere la velocità max, dunque il robot viaggia ad una velocità molto inferiore.

Per questa piattaforma è stato cercato di raggiungere l'equilibrio di una frequenza più alta possibile, ma con valori di velocità accettabili per applicazioni che considerano l'intero spazio di lavoro.

Guardando invece le prove statistiche effettuate, come già specificato nel capitolo inerente le prove non sono sufficienti a descrivere in maniera ottimale il comportamento del robot ma forniscono un ottima base per partire ed effettuare prove più specifiche: oltre al quadrato effettuato sarebbe bene provare percorsi più spezzati e alcuni più curvilinei in modo da guardare tutte le possibilità di movimento. Inoltre è sempre stata utilizzata la funzione PTP, quindi considerare il comportamento di Line e PLine è un obbiettivo futuro.

Bibliografia

[1] www.robots.ieee.org.

[2] www.researchgate.net.

[3] www.aapm.org.

[4] www.marketsandmarkets.com.

[5] J. J. Craig, *Introduction to Robotics*. Usa: Pearosn Education, 2005.

[6] R. S. Andersen, "Kinematics of a ur5," 2018.

[7] T. R. inc., *TM5 Guide Book*, 2 2018.

[8] www.tm robot.com.

[9] T. R. inc., *Expression Editor and Listen Node*, 12 2019.

[10] ——, *Software Manual TMflow*, 5 2020.

[11] G. R. Aldo Rossi, Paolo Gallina and V. Zanotto, "Rate-to-force control in admittance mode bilateral teleoperation," *China MachineryPress,*, 2003.

[12] O. K. Bruno Siciliano, *Springer Handbook of Robotics*. Springer-Verlag Berlin Heidelberg: Springer, 2016.

Ringraziamenti

Desidero ringraziare il Professor Rosati per i preziosi insegnamenti durante il percorso accademico e la ricerca della tesi.
Ringrazio l'Ingegner Matteo Bottin per la disponibilità e il prezioso aiuto nello sviluppo del lavoro in laboratorio.
Ringrazio la mia famiglia e tutti gli amici per avermi accompagnato in questi anni universitari, senza di loro non sarebbero stati gli stessi.