



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN  
INGEGNERIA INFORMATICA

# Analisi dei sensori coppia-forza per lo sviluppo di applicazioni industriali in ROS

*Relatore:*

PROF. STEFANO GHIDONI

*Correlatore:*

MATTEO TERRERAN, PhD

*Laureando:*

ANDREA STOCCO

2009353

Anno Accademico 2022/2023



## **Abstract**

I sensori coppia-forza sono componenti fondamentali nei sistemi robotici in quanto forniscono dati sulle forze e i momenti esterni applicati al robot. Questi dati possono essere utilizzati per applicazioni a supporto della collaborazione uomo-robot e per l'automatizzazione di attività che richiedono elevata precisione. ROS (Robot Operating System) é un framework che fornisce una vasta gamma di librerie e strumenti software per lo sviluppo di applicazioni robotiche. In questa tesi verranno mostrate delle possibili applicazioni ROS in ambito industriale per i sensori coppia-forza, previa verifica della loro accuratezza in termini di reattività e precisione. Questa verifica sarà effettuata attraverso due esperimenti: il primo riguardante il taglio di un filo a cui é attaccato un peso e il secondo relativo al calcolo della viscosità di un fluido. I risultati ottenuti da tali esperimenti forniranno una solida base di validazione per l'utilizzo dei sensori coppia-forza nelle applicazioni industriali, dimostrando la loro capacità di rispondere tempestivamente ai cambiamenti delle forze in gioco e di fornire misurazioni precise.



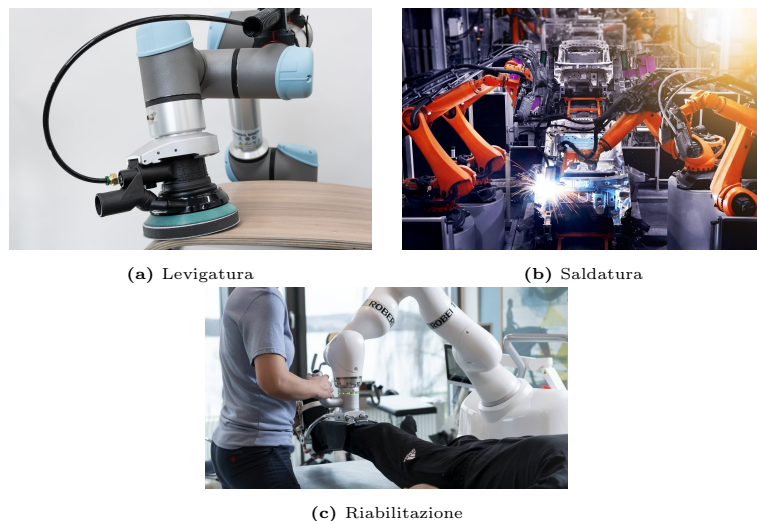
# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 ROS</b>	<b>3</b>
1.1 Nodo . . . . .	3
1.2 Topic . . . . .	4
1.3 Service . . . . .	5
1.4 Topic e Service a confronto . . . . .	5
1.5 Workspace . . . . .	6
<b>2 Ambiente di lavoro</b>	<b>7</b>
2.1 UR5 . . . . .	7
2.2 FT300-S . . . . .	9
2.2.1 Collegamento via USB tra sensore e control box e via ether-	
net tra control box e computer . . . . .	10
2.2.2 Collegamento diretto via USB tra sensore e computer . . . .	11
2.3 MoveIt . . . . .	12
2.4 Controllori . . . . .	13
<b>3 Validazione del sensore</b>	<b>15</b>
3.1 Errore nelle misurazioni . . . . .	15
3.2 Taglio del filo . . . . .	16
3.3 Calcolo della viscosità . . . . .	17
<b>4 Applicazioni industriali</b>	<b>19</b>
4.1 Inseguitore di forza . . . . .	19
4.2 Pick and place . . . . .	20
4.2.1 Salvataggio delle posizioni . . . . .	20
4.2.2 Posizionamento . . . . .	21
4.2.3 Inserimento . . . . .	21

4.3	Movimento a spirale . . . . .	21
4.4	Trasporto collaborativo . . . . .	22
<b>5</b>	<b>Conclusioni</b>	<b>25</b>
	<b>Bibliografia</b>	<b>27</b>

# Introduzione

I robot manipolatori collaborativi hanno rivoluzionato l'automazione industriale, permettendo lo svolgimento di operazioni complesse in modo rapido, preciso e sicuro. La loro importanza nel campo dell'automazione industriale e di altre applicazioni é in continua crescita, con sempre piú settori che ne riconoscono il valore e ne adottano l'utilizzo. Un ruolo chiave nel controllo di questi robot viene assunto dai sensori coppia-forza, che permettono di misurare e regolare la forza esercitata dal robot durante lo svolgimento delle proprie attività. La loro versatilità li rende strumenti preziosi in molti campi della **robotica industriale** e della **medicina**. Essi infatti consentono al robot di controllare la forza esercitata durante operazioni di assemblaggio, levigatura, saldatura o manipolazione degli oggetti, come mostrato in Figura 1. Vengono inoltre utilizzati nella riabilitazione fisioterapica, per valutare la forza muscolare e i progressi del paziente.



**Figura 1:** Applicazioni dei sensori coppia-forza

Questi sensori sono in grado di convertire le forze e le coppie applicati ad essi in segnali elettrici che possono essere interpretati da altri dispositivi. Esistono varie tipologie di sensori coppia-forza, ognuna delle quali ha un diverso

meccanismo di funzionamento. I sensori **piezoelettrici**, per esempio, sfruttano la proprietà di alcuni materiali (cristalli piezoelettrici) di generare una carica elettrica se sottoposti a deformazione meccanica. Tale variazione può essere misurata per determinare la forza o la coppia applicata. Il sensore **FT 300-S** di **Robotiq** sfrutta proprio questo principio di funzionamento ed è in grado di misurare forze e coppie lungo i sei gradi di libertà (x, y, z, roll, pitch, yaw). Uno dei modelli di robot manipolatori più utilizzato è l'**UR5** di **Universal Robot**. Si è scelto di utilizzarlo in questo studio, per via della sua flessibilità ed efficienza. L'UR5, a differenza di altri robot collaborativi, non è provvisto di sensori coppia-forza integrati. È stato dunque necessario installare l'FT 300-S manualmente. In questa tesi verrà presentata l'implementazione di un sistema di controllo della forza per l'UR5 utilizzando i dati forniti dall'FT 300-S e il framework di sviluppo **ROS (Robot Operating System)**. ROS è un framework ampiamente utilizzato dalla comunità informatica perché fornisce strumenti e librerie per il controllo e la comunicazione tra le componenti di un sistema robotico. Inizialmente verrà presentata una panoramica sui concetti fondamentali di ROS e sul sensore coppia-forza, mostrando anche due diverse modalità di interfacciamento con esso (vedi Capitoli 1 e 2). Nel Capitolo 3 verrà, invece, riportata un'analisi per la valutazione delle prestazioni del sensore in termini di reattività e precisione. Infine, sarà presentato il setup sperimentale (vedi Capitolo 4) necessario allo svolgimento delle applicazioni presentate nel Capitolo 5 volte a dimostrare l'efficacia di tali sensori per lo svolgimento di attività industriali, come il **pick and place** e il **trasporto collaborativo**.



# Capitolo 1

## ROS

ROS (Robot Operating System) é un framework open-source disponibile in Python e C++ per lo sviluppo di applicazioni robotiche. Si tratta di un sistema centralizzato che permette alle diverse componenti del sistema (nodi) di comunicare tra loro sia in modo asincrono (topic) che sincrono (service). Offre inoltre una vasta gamma di strumenti di sviluppo, come un software per la visualizzazione grafica (RViz) e la possibilità di registrare e riprodurre dati, favorendo così il debugging e il testing delle applicazioni. La versione raccomandata e utilizzata é **ROS Noetic** per **Ubuntu Focal 20.04**. Questo capitolo ci fornirà una panoramica esaustiva dei concetti base del Robot Operating System e dell'ambiente di esecuzione associato [2].

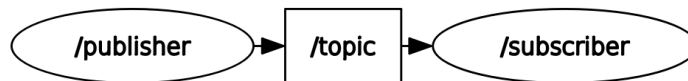
### 1.1 Nodo

Un **pacchetto** ROS contiene al suo interno codici sorgenti, librerie e dati di configurazione. Per creare un pacchetto é sufficiente eseguire il comando `catkin_create_pkg <nome_pacchetto>` all'interno della cartella `src/` situata nel workspace catkin. Un **nodo** é un eseguibile che sfrutta ROS per comunicare con altri nodi. Quando viene lanciato il comando `catkin_make`, ogni file sorgente in ogni pacchetto viene compilato dando origine ad un nodo. Impropriamente, si potrebbe quindi dire che un pacchetto é un insieme di nodi riguardanti la stessa applicazione. Per poter eseguire un nodo é sufficiente eseguire il comando `roslaunch <nome_pacchetto> <nome_nodo>`. Prima di eseguire un nodo é necessario, tuttavia, avviare un ROS Master. Lo scopo principale di un **ROS Master** é quello di consentire ai singoli nodi di localizzarsi a vicenda. Una volta fatto par-

tire, essi potranno comunicare tra loro attraverso topic o service. Per eseguire un ROS Master sarà sufficiente eseguire il comando `roscore` in un altro terminale.

## 1.2 Topic

I **topic** sono dei canali di comunicazione unidirezionali che consentono lo scambio di informazioni tra nodi sottoforma di messaggi. Un nodo che pubblica messaggi su un topic viene chiamato **publisher**, mentre un nodo che legge i messaggi da un topic viene chiamato **subscriber**.



**Figura 1.1:** Schema di comunicazione

Per pubblicare un messaggio su un topic bisogna utilizzare l'apposita funzione `publish()` passandole come parametro il messaggio che vogliamo pubblicare. A discapito del nome, questa funzione non pubblica effettivamente il messaggio, ma lo mette in una coda d'attesa (la cui dimensione viene specificata quando viene istanziato il publisher). Un thread separato si occupa di inviare effettivamente il messaggio al topic per renderlo visibile a tutti i nodi subscriber connessi. Se il numero di messaggi in coda supera la sua dimensione, i messaggi più vecchi verranno cancellati per fare spazio a quelli più recenti. Quando arriva un nuovo messaggio al subscriber, esso viene salvato in una coda d'attesa (stesso funzionamento di quella del publisher) fino a quando ROS non dá la possibilità al nodo di eseguire la funzione di callback. Tale funzione é definita dall'utente e si occupa di processare il messaggio ricevuto. ROS eseguirá una callback solo quando gli verrà dato il permesso di farlo. Ci sono due modi per farlo:

- `ros::spinOnce()` chiede a ROS di eseguire tutte le callback in sospenso restituendo poi il controllo all'utente
- `ros::spin()` chiede a ROS di attendere e di eseguire tutte le callback in sospenso fino a quando il nodo non viene spento. É equivalente a:

```
while (ros::ok()) {  
    ros::spinOnce();  
}
```

`ros::ok()` ritorna 0 quando:

- il nodo viene spento attraverso un SIGINT (Ctrl-C) oppure dalla chiamata di `ros::shutdown()` in un altro punto del codice
- un altro nodo con lo stesso nome viene eseguito

In altre parole `ros::spin()` vincola il nodo a rimanere sempre e solo in attesa di leggere nuovi messaggi. Se il nodo non deve solamente eseguire le callback, allora un loop con `ros::spinOnce()` é la scelta corretta.

## 1.3 Service

I **service** sono un'altra modalità di comunicazione tra nodi. A differenza dei topic, che consentono la comunicazione asincrona, i service instaurano una comunicazione di tipo 'client-server'. Il nodo **client** invia una richiesta al nodo service e attende la sua risposta prima di andare avanti con l'esecuzione. Per implementare un service in ROS é necessario per prima cosa definire la struttura dei messaggi di richiesta e risposta [6]. Successivamente il client potrà creare una richiesta nel formato specificato e inviarla al nodo service come parametro della funzione `call()`. Il nodo service elaborerà quindi la richiesta fornendo una risposta al client.

## 1.4 Topic e Service a confronto

I topic e i service sono due modalità di comunicazione molto diverse a livello concettuale. Con un topic si instaura una comunicazione asincrona in cui tutti i subscriber connessi attendono la pubblicazione di un messaggio da parte del publisher, mentre con un service é il 'server' che rimane in attesa di richieste da parte dei client connessi. Questo rende i topic più adatti per la trasmissione di flussi di dati continui (come quelli provenienti dai sensori) e i service più indicati nel caso di servizi puntuali, come la richiesta di un calcolo o di un'altra specifica azione ad un altro nodo [5]. Nel corso di questa tesi verranno utilizzate entrambe le modalità di comunicazione, con prevalenza di quella topic.

## 1.5 Workspace

Per poter eseguire codice ROS serve un ambiente che permetta l'organizzazione e l'utilizzo di tutti i pacchetti necessari. Al riguardo, **catkin** é il sistema di compilazione ufficiale di ROS che consente la creazione di un workspace per organizzare e gestire le applicazioni. I termini 'pacchetto' e 'applicazione' sono interscambiabili e possono essere utilizzati in modo equivalente. Una volta creato il workspace catkin [3], il codice sorgente contenuto all'interno dei pacchetti potrà essere compilato ed eseguito.

# Capitolo 2

## Ambiente di lavoro

Questo capitolo offrirá una panoramica sul setup dell'ambiente di lavoro. Si parlerá diffusamente delle caratteristiche e delle specifiche tecniche del robot e del sensore. Verranno, inoltre, mostrati i passaggi fondamentali che hanno consentito il controllo del sistema attraverso ROS.

### 2.1 UR5

La versione attuale dell'UR5 é quella appartenente alla **e-series**, rilasciata nel 2018. Tuttavia, in questa tesi, viene utilizzata la versione precedente della famiglia **CB3**, commercializzata a partire dal 2008.

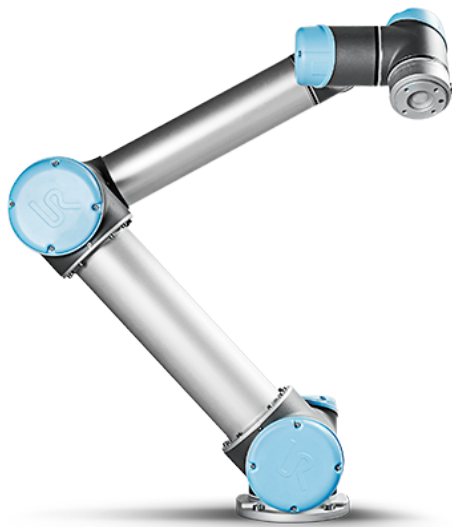
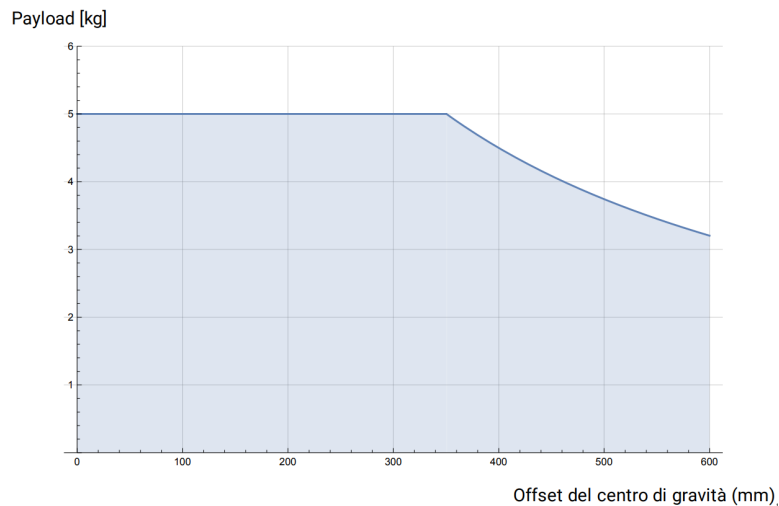


Figura 2.1: UR5/CB3

All'estremità del robot é possibile installare un **end effector**, ossia un dispositivo concepito per la manipolazione degli oggetti che fornisce l'unica possibile interazione con l'ambiente esterno. Il carico massimo sopportabile dall'UR5 dipende dall'offset del centro di gravità, ossia la distanza tra l'estremità del braccio robotico (punto di applicazione dell'end effector) e il centro di gravità dell'UR5.



**Figura 2.2:** Andamento del carico massimo sopportabile rispetto all'offset del centro di gravità

Come si può notare in Figura 2.2, l'UR5 é in grado di sopportare un carico massimo di 5kg finché l'estensione del braccio non supera i 350mm. Da qui in poi al crescere dell'offset seguirá una diminuzione del carico massimo applicabile. L'UR5 é collegato ad una **control box** che é un dispositivo alimentato elettricamente in grado di fornire energia sia al robot che alle periferiche collegate ad esso. Sulla control box é presente poi un'uscita ethernet per la connessione ad un PC per il controllo remoto, da cui vengono lanciati tutti gli applicativi sviluppati.



(a) Control box



(b) Teach pendant

**Figura 2.3**

In Figura 2.3 vengono mostrati la control box e il **Teach Pendant**, un dispositivo touchscreen collegato alla control box che consente il controllo diretto del robot. Sul Teach Pendant é eseguita un'interfaccia chiamata **Polyscope** che permette la programmazione dei movimenti del robot e il cambiamento di alcune sue impostazioni. Per poter controllare l'UR5 tramite ROS é, tuttavia, necessario installare sul Teach Pendant il plugin **externalcontrol.urcap**, creare un nuovo programma su Polyscope che preveda l'utilizzo del plugin e impostare l'indirizzo ip del computer remoto da cui verranno lanciati i nodi ROS.

## 2.2 FT300-S

Il sensore FT300-S di Robotiq é stato installato all'estremitá dell'UR5 e collegato alla control box tramite il proprio cavo di alimentazione.



Figura 2.4: FT300-S

É importante notare come la sua presenza non precluda la possibilità di installazione anche di un end effector, che può essere facilmente posizionato 'al di sopra' del sensore. L'FT300-S é in grado di rilevare forze e torsioni nel range di, rispettivamente,  $\pm 300N$  e  $\pm 30Nm$ . Siccome le misurazioni del sensore hanno un rumore di fondo intrinseco, é necessario scartare tutti i dati al di sotto delle soglie consigliate nel manuale [8] in quanto non attendibili. Per interfacciarsi con il sensore dal PC sono disponibili due modalità di comunicazione: **ModbusRTU** e **data stream**. La prima viene utilizzata per inviare comandi al sensore (es. azzeramento) e per richiedere informazioni su di esso, la seconda per ottenere un flusso continuo di dati relativi alle misurazioni effettuate. Per usufruire di tali modalità di comunicazione sono state provate due alternative:

- collegamento via USB tra sensore e control box e via ethernet tra control box e computer

- collegamento diretto via USB tra sensore e computer

### 2.2.1 Collegamento via USB tra sensore e control box e via ethernet tra control box e computer



**Figura 2.5:** Schema collegamento

Con la configurazione mostrata in Figura 2.5, per poter leggere i dati provenienti dal sensore, é stato necessario sviluppare un **driver**. Poi si é stabilita una connessione di rete tramite un **socket** collegato all'indirizzo IP del robot, alla porta **63351**. Su tale porta il sensore invierà un flusso continuo di messaggi ad una frequenza di 100Hz [8]. Come da manuale, i messaggi sono lunghi 16 byte e hanno la seguente struttura:

```

buff[0] = 0x20
buff[1] = 0x4E
buff[2] = Fx * 100 (LSB)      LSB = Least Significant Bit
buff[3] = Fx * 100 (MSB)      MSB = Most Significant Bit
buff[4] = Fy * 100 (LSB)
buff[5] = Fy * 100 (MSB)
buff[6] = Fz * 100 (LSB)
buff[7] = Fz * 100 (MSB)
buff[8] = Mx * 1000 (LSB)
buff[9] = Mx * 1000 (MSB)
buff[10] = My * 1000 (LSB)
buff[11] = My * 1000 (MSB)
buff[12] = Mz * 1000 (LSB)
buff[13] = Mz * 1000 (MSB)
  
```



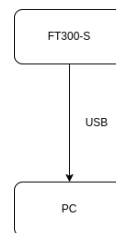
```
buff[14] = LSB CRC      CRC = Cyclic Redundancy Check
buff[15] = MSB CRC
```

Ogni elemento dell'array contiene un byte in formato esadecimale. Le forze (F), i momenti (M) e il CRC vengono rappresentati con 2 byte ciascuno. Il byte più significativo e quello meno significativo vengono divisi e inviati come elementi differenti. I primi due byte sono fissati, gli ultimi due rappresentano il CRC (che consente la rilevazione di eventuali errori di trasmissione) e quelli intermedi codificano le forze e i momenti percepiti dal sensore. La control box, riceve tali messaggi e li converte nel formato:

(Fx, Fy, Fz, Mx, My, Mz)

Per rendere disponibili i dati ricevuti agli altri nodi, il driver, dopo averli convertiti in decimale, li pubblica sul topic `sensor_topic` [9].

### 2.2.2 Collegamento diretto via USB tra sensore e computer

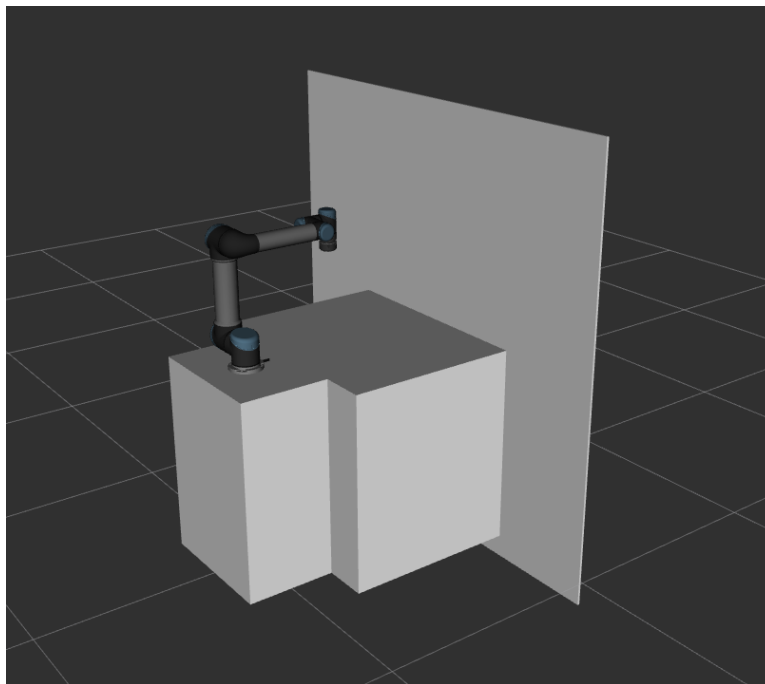


**Figura 2.6:** Schema collegamento

Con questa configurazione, invece, un driver per la lettura dei dati del sensore ci viene già fornito da Robotiq. Una volta scaricato il loro repository GitHub [10], per eseguire il driver è sufficiente far partire un nuovo nodo ROS con il comando `roslaunch robotiq_ft_sensor rg_sensor`. Il driver consente l'utilizzo di entrambe le modalità di comunicazione descritte in precedenza (ModbusRTU e data stream). Viene infatti creato il service `robotiq_ft_sensor_acc` per l'invio di comandi al sensore come, ad esempio, la richiesta di azzeramento. Inoltre, viene generato il topic `robotiq_ft_wrench` in cui vengono pubblicate le misurazioni prodotte dal sensore. Quindi, per avere il pieno controllo del sensore, è sufficiente creare un client per le richieste al service `robotiq_ft_sensor_acc` e un subscriber per leggere i dati presenti sul topic `robotiq_ft_wrench`. Per gli esperimenti e le applicazioni industriali descritte nel Capitoli 3 e 4 si è scelto di utilizzare questo secondo approccio e quindi di collegare direttamente il sensore al PC via USB.

## 2.3 MoveIt

Dopo aver introdotto i concetti base di ROS e descritto le principali caratteristiche di robot e sensore, in questa sezione si parlerà di **MoveIt**. MoveIt é un framework specifico di ROS specializzato nella **pianificazione del movimento**. Offre un'ampia gamma di strumenti e librerie per la generazione delle traiettorie, la gestione della cinematica, la simulazione e il controllo dei robot. Prima di procedere, é però opportuno fornire una breve introduzione ai file URDF. Gli **URDF (Unified Robot Description Format)** sono dei file basati sul formato XML (eXtensible Markup Language) e costituiscono uno standard per rappresentare la geometria, la cinematica e altre caratteristiche dei robot all'interno di ROS. Grazie a questi file, è possibile definire la gerarchia dei **link** del robot, specificandone anche informazioni quali le dimensioni, la massa e l'inerzia. Inoltre, gli URDF consentono di modellare i **giunti** del robot, definendone i limiti di movimento e le relazioni cinematiche con i link adiacenti. Robotiq e Universal Robot mettono a disposizione i file URDF dei propri prodotti. Per ricreare l'ambiente di lavoro presente in laboratorio é stato necessario 'unire' la rappresentazione del robot con quella del sensore in un nuovo file URDF contenente anche caratteristiche proprie dell'ambiente, come il tavolo su cui é montato il braccio, il piano di lavoro e il muro.

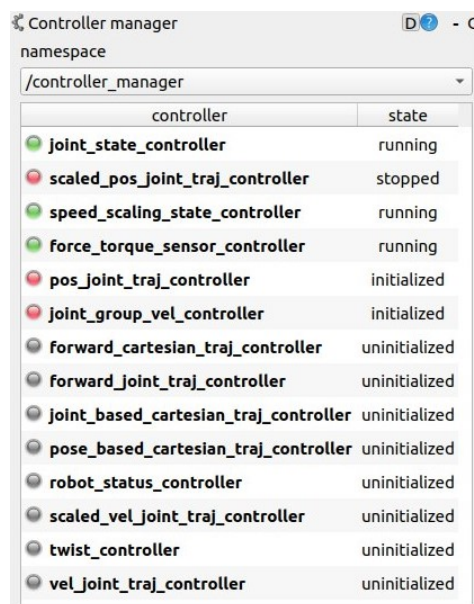


**Figura 2.7:** Simulazione dell'ambiente di lavoro su RViz

In Figura 2.7 viene mostrato l'ambiente di lavoro in cui sono state provate le applicazioni proposte. **RViz (ROS Visualization)** é uno strumento di visualizzazione 3D incluso in ROS che, oltre a consentire la visualizzazione dei movimenti del robot, offre altri strumenti per interagire con esso. Per semplificare il processo di configurazione e setup del sistema, MoveIt mette a disposizione il **MoveIt Setup Assistant**, un software che fornisce un'interfaccia grafica per permettere agli utenti di generare i file di configurazione necessari all'utilizzo di MoveIt. In [12] la cartella `ur5_ft_moveit_config` é stata generata da MoveIt Setup Assistant a partire dal file `ur5_ft.urdf.xacro` (contenente la descrizione dell'ambiente) presente all'interno di `environment_description`. In `environment_manager` sono presenti dei **launch file** (file XML per l'avvio simultaneo di piú nodi, che permettono la definizione e il passaggio di parametri tra di essi) che a cascata fanno partire nodi per il collegamento remoto del PC al robot, l'avvio di MoveIt e Rviz e l'inizializzazione del sensore. Sarà dunque sufficiente eseguire il comando `roslaunch environment_manager ur5_ft_load_all` per essere poi in grado di eseguire le applicazioni proposte nel Capitolo 4.

## 2.4 Controllori

L'UR5 possiede diversi **controllori** integrati per gestire il movimento e il funzionamento del robot.



The screenshot shows the 'Controller manager' window in ROS. It displays a list of controllers and their current states. The namespace is set to '/controller\_manager'.

controller	state
joint_state_controller	running
scaled_pos_joint_traj_controller	stopped
speed_scaling_state_controller	running
force_torque_sensor_controller	running
pos_joint_traj_controller	initialized
joint_group_vel_controller	initialized
forward_cartesian_traj_controller	uninitialized
forward_joint_traj_controller	uninitialized
joint_based_cartesian_traj_controller	uninitialized
pose_based_cartesian_traj_controller	uninitialized
robot_status_controller	uninitialized
scaled_vel_joint_traj_controller	uninitialized
twist_controller	uninitialized
vel_joint_traj_controller	uninitialized

Figura 2.8: Lista controllori UR5

In Figura 2.8, viene mostrata la lista dei controllori disponibili per manovrare l'UR5 insieme ai rispettivi stati di funzionamento. I controllori segnati in verde sono Read-Only controllers, ossia dei controllori che leggono solamente lo stato attuale del robot e lo pubblicano su un topic. Non c'è nessuna limitazione sul numero di controllori Read-Only in esecuzione nello stesso momento. Gli altri, invece, sono Commanding controllers, ossia controllori che consentono l'alterazione dello stato del robot. Non è possibile utilizzare più controllori di questo tipo contemporaneamente. MoveIt permette un controllo di tipo **posizionale**, ossia, internamente, calcola la traiettoria migliore per andare da un punto di partenza nello spazio ad un punto di arrivo. Una volta calcolata la traiettoria, con l'ausilio di `scaled_pos_joint_traj_controller` vengono modificati i valori dei giunti per consentire al robot di seguire la traiettoria specificata. Nelle applicazioni mostrate nel Capitolo 4 viene utilizzato anche il controllore di **velocità** `twist_controller`. Questo controllore riceve un comando di twist come input, che specifica la velocità di traslazione e rotazione lungo gli assi x, y e z. Il controllore, poi, lo traduce in comandi di controllo per i motori del robot. Essendo entrambi Commanding controllers, l'utilizzo di uno esclude l'utilizzo dell'altro. ROS mette a disposizione dei service per gestire i controllori attivi del robot. Ad esempio, `/controller_manager/switch_controller` si occupa di scambiare lo stato di esecuzione dei due controllori che riceve in input [13]. Tali service risultano, quindi, molto utili nelle applicazioni proposte perché permettono l'utilizzo del controllore più appropriato in base alle specifiche esigenze.

## Capitolo 3

# Validazione del sensore

In questo capitolo verranno mostrati degli esperimenti per valutare il funzionamento e le prestazioni del sensore. Per l'analisi della **reattività**, viene osservato il comportamento del sensore nel caso in cui ci sia un cambiamento istantaneo delle forze in gioco. Un'altro importante aspetto da valutare é la **precisione** dei dati forniti dal sensore. Per farlo si é pensato di utilizzare le misurazioni effettuate per calcolare la viscosità di un liquido di cui se ne conosce il valore. Prima di mostrare i risultati di questi due esperimenti é bene, però, parlare dell'importanza dell'azzeramento periodico del sensore.

### 3.1 Errore nelle misurazioni

Come spiegato nel Capitolo 2 il sensore é soggetto a rumore di fondo intrinseco, che può essere causato da diversi fattori, come la temperatura, la stabilità dell'alimentazione o il rumore elettrico.

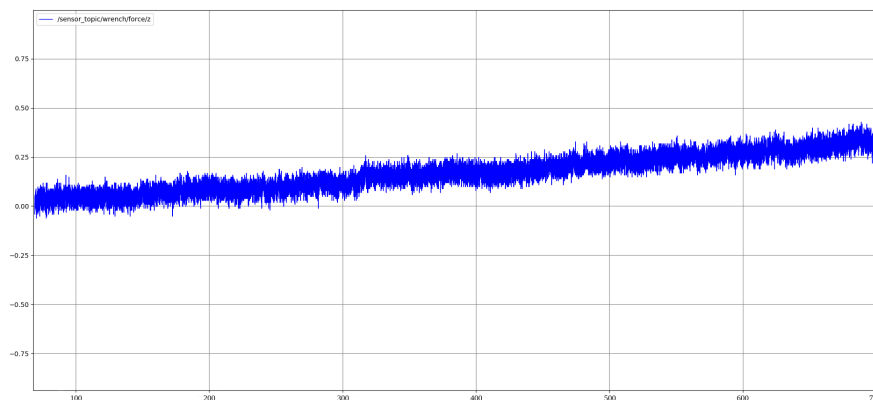
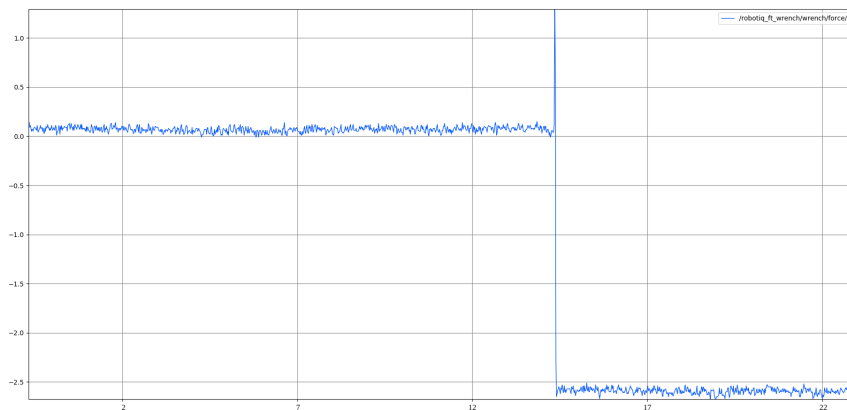


Figura 3.1: Drifting lungo l'asse z

In Figura 3.1 viene mostrato il fenomeno del **drifting**: ossia quando un sensore nel corso del tempo mostra una deviazione nelle sue letture senza un'effettiva variazione delle condizioni ambientali. Si può notare come la forza rilevata dal sensore lungo l'asse z tenda a crescere col passare del tempo, senza che al sensore venga applicata alcuna forza. Già dopo 200 secondi, la forza misurata supera la soglia di confidenza specificata nel manuale entro la quale la misurazione deve essere catalogata come non attendibile. Per ovviare a questo problema è necessario azzerare il sensore periodicamente, in modo che le letture risultino corrette e senza deviazioni.

## 3.2 Taglio del filo

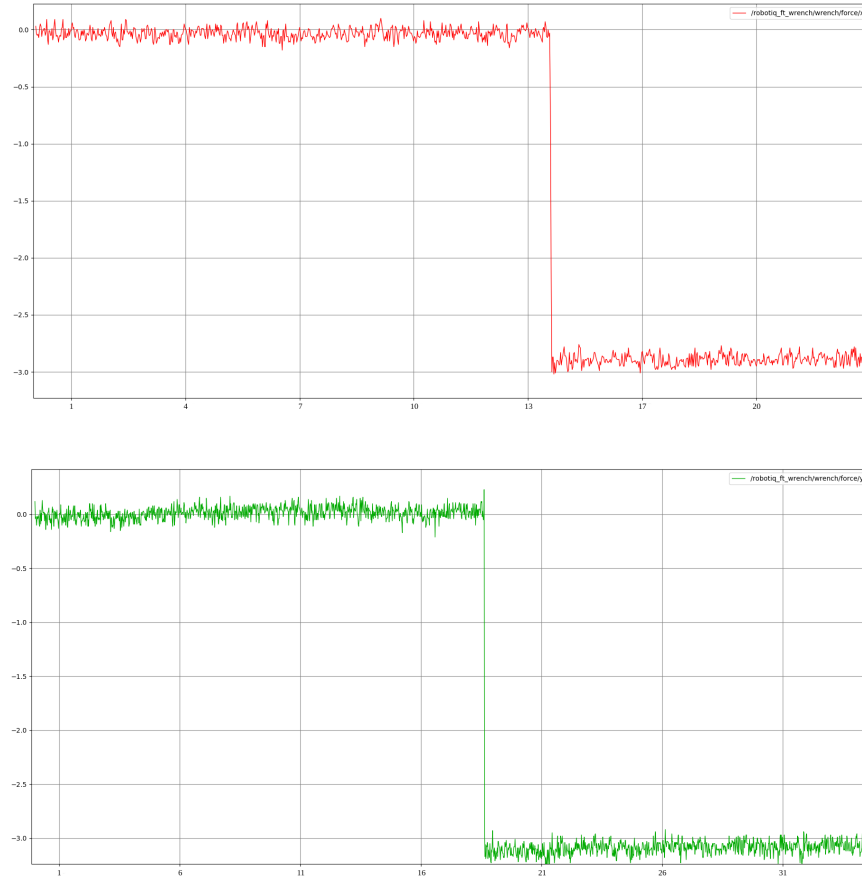
In questo esperimento, è stato attaccato al sensore un filo con appeso un oggetto di 0.25Kg. Il braccio è stato posizionato in modo tale che la forza peso gravasse solo su un asse del sensore alla volta. Dopo aver azzerato il sensore, per verificarne la reattività, il filo è stato tagliato di netto. Il taglio del filo è un ottimo modo per 'simulare' un cambiamento di forza istantaneo.



**Figura 3.2:** Andamento taglio del filo lungo l'asse z

In Figura 3.2 viene mostrato l'andamento della forza rilevata dal sensore lungo l'asse z. Si può notare che, fino a quando il filo è attaccato al sensore, la forza rilevata è circa zero. Questo perché il sensore è stato azzerato quando l'oggetto era già stato appeso. Dopo circa 15 secondi, il filo viene tagliato. In questo istante il sensore rileva per una frazione di secondo una forza di circa 1.5N (probabilmente dovuta ad un taglio non sufficientemente netto), per poi assestarsi al valore reale della forza rilevata, ossia circa -2.5N. Tale esperimento è stato ripetuto anche per

gli altri due assi con esiti leggermente migliori. I risultati vengono mostrati in Figura 3.3.



**Figura 3.3:** Andamento esperimento lungo x e y

### 3.3 Calcolo della viscosità

In questa sezione viene mostrato un esperimento per valutare la precisione delle misurazioni del sensore. Per farlo si é pensato di usare i valori delle forze misurate per calcolare la **viscosità** del burro d’arachidi, che tipicamente é compresa tra  $1500\text{--}2500\text{ Pa}\cdot\text{s}$ . A tal proposito é stato installato sull’UR5 una spatola rettangolare (di dimensioni  $6\text{ cm} \times 4\text{ cm} \times 4\text{ mm}$ ) come end effector. La spatola, una volta immersa nel burro di arachidi, viene fatta ruotare attorno al suo asse con velocità angolare costante. Il sensore, rileva quindi un momento torcente lungo l’asse z corrispondente alla forza di attrito viscoso esercitata dal fluido sulla spatola in rotazione. É dunque possibile utilizzare tali valori per ricavare sperimentalmente la viscosità del burro d’arachidi e confrontarla con i dati ufficiali noti. La formula

per il calcolo dell'attrito viscoso é la seguente

$$F = 2 \cdot h \cdot l \cdot \eta \cdot \omega$$

con

- $h$ : altezza della porzione di spatola immersa nel fluido
- $l$ : larghezza della spatola
- $\eta$ : viscosità del fluido
- $\omega$ : velocità angolare di rotazione

Si può quindi ‘ribaltare’ tale formula per ricavare la viscosità dalla forza di attrito misurata

$$\eta = \frac{F}{2 \cdot h \cdot l \cdot \omega} \quad (3.1)$$

In [14] viene mostrato il codice ROS per effettuare l'esperimento. Con MoveIt il braccio viene prima posizionato in modo tale che la spatola sia immersa all'interno del burro d'arachidi. Come indicato in 3.1, per calcolare la viscosità del fluido é necessario conoscere la velocità di rotazione della spatola. Con un controllore di posizione, tale informazione non é accessibile. Pertanto é necessario passare ad un controllore di velocità (quale `twist_controller`) per poter manovrare il robot in termini di velocità e non in termini di posizione. Essendo questa un'operazione effettuata anche in altre applicazioni (vedi Capitolo 4), sono state create delle apposite funzioni per il cambio dei controllori nel file `utils.cpp`. Il robot, quindi, comincia a far ruotare su se stessa la spatola a velocità costante mentre il sensore acquisisce i dati della forza d'attrito. Viene poi calcolata la media di tutte le misurazioni effettuate (600) e , tale valore, viene usato per calcolare la viscosità del burro d'arachidi. Il risultato é  $\eta = 1998.0468 \text{ Pa} \cdot \text{s}$ , che rientra nel range di valori noto specificato inizialmente. Le misurazioni effettuate dal sensore sono, quindi, precise e affidabili.



## Capitolo 4

# Applicazioni industriali

Dopo aver validato la reattività e la precisione del sensore, in questo capitolo verranno mostrate delle possibili applicazioni dei sensori coppia-forza in ambito industriale. La prima applicazione che verrà trattata è un ‘inseguitore di forze’ che consente ad un operatore di muovere il braccio a piacere, senza dover utilizzare il teach pendant. Successivamente verrà mostrato come tale applicazione possa essere versatilmente adattata anche per altri scopi. Infatti, è fondamentale nel task di presa e posizionamento per indicare al braccio la posizione da raggiungere per completare con successo il compito assegnato, oltre che di cruciale importanza nel trasporto collaborativo uomo-robot.

### 4.1 Inseguitore di forza

Come accennato, questa applicazione si focalizza sulla necessità di poter muovere il robot a piacimento, senza dover ricorrere all'utilizzo del teach pendant o RViz. Diventa di fondamentale importanza se integrata in applicazioni collaborative, per rendere più user friendly l'interfacciamento con il robot. Quando il sensore rileva delle forze superiori ad una determinata soglia (calcolata sperimentalmente), viene inviato al robot un comando **Twist** per farlo muovere con una velocità proporzionale alla forza applicata in input. In questo modo, il robot ‘segue’ le forze impartite dall'operatore convertendole in termini di velocità di movimento. Il codice ROS che implementa tale funzionalità viene mostrato in [15]. Dopo aver portato il robot in posizione di partenza con MoveIt, viene attivato `twist_controller` con la stessa funzione utilizzata nell'esperimento del burro d'arachidi (presente in `utils.cpp`). Per il momento non è necessario focalizzarsi sulla funzione `feedback()`, sul publisher `gripper_publisher` e nemmeno

sul vettore **positions**, in quanto sono elementi che verranno utilizzati nell'applicazione **pick and place**. All'interno del vettore **forces** vengono salvate le ultime 20 misurazioni del sensore. Di queste ne viene calcolata la media e, se il modulo é superiore alla soglia specificata, viene inviato al robot un comando **Twist** contenente uno scalamento delle forze rilevate nelle tre componenti. Il calcolo della velocità di movimento viene fatto sulla media degli ultimi campioni per una questione di utilizzabilità. Servirsi delle misurazioni singole per il calcolo del vettore velocità da inviare al robot porta ad un movimento poco fluido e impulsivo che peggiora l'esperienza utente. La conversione tra forza e velocità avviene mediante un'attenuazione delle componenti del vettore media per un coefficiente costante. Questo per ridurre l'influenza delle forze 'piccole' nel vettore velocità risultante, valorizzando maggiormente le componenti più ampie.

## 4.2 Pick and place

Il pick and place é un'applicazione largamente utilizzata in ambito industriale e consiste nello spostamento di un oggetto da un punto di partenza ad uno di destinazione. Tale compito può essere portato a termine senza l'ausilio di un sensore coppia-forza. L'alternativa proposta in questa tesi utilizza i dati forniti dal sensore per migliorare l'interfacciamento con il robot e la precisione nel posizionamento dell'oggetto. Per fare ciò é stato necessario installare un gripper come end effector dell'UR5 [?]. In [12] con MoveIt Setup Assistant é stata generata la cartella contenente tutti i file di configurazione per questo specifico setup. In Figura ?? viene mostrato l'ambiente di lavoro comprensivo del gripper per la presa degli oggetti. Di seguito verranno descritte le parti in cui é stata suddivisa l'applicazione.

### 4.2.1 Salvataggio delle posizioni

Inizialmente, viene utilizzato l'**inseguitore di forza** per il salvataggio delle posizioni di partenza e di destinazione su cui il robot dovrà spostarsi per portare a termine il proprio compito. L'operatore potrà, quindi, muovere il braccio liberamente fino a quando non si trova al di sopra dell'oggetto da spostare. Sarà, poi, sufficiente applicare una piccola torsione al sensore affinché la posizione venga salvata nel vettore **positions**. In caso di successo, il gripper si aprirà e si chiuderà velocemente per dare all'utente un feedback visivo. Lo stesso dovrà essere fatto anche per la posizione di destinazione e per una posizione fittizia in una zona

libera del piano di lavoro (vedi 4.2.2). Una volta terminata la fase di acquisizione delle posizioni, le prime due verranno pubblicate sul topic `task_positions`.

### 4.2.2 Posizionamento

In questa fase, il nodo **placement** [16] potrà iscriversi al topic e leggere le posizioni precedentemente acquisite. Inizialmente, dalla posizione fittizia, il robot scenderá verso il basso fino a quando non verrà rilevata una forza lungo l'asse *z*, corrispondente al contatto con il piano di lavoro. Tale posizione verrà salvata e utilizzata in seguito. Con MoveIt il braccio viene spostato alla prima posizione del topic, ossia il punto in cui é presente l'oggetto da prendere. Da qui, si muoverá verso il basso alla ricerca di un contatto con esso e una nuova posizione verrà salvata. Le differenza delle due posizioni corrisponde all'altezza dell'oggetto. Sarà, dunque, sufficiente chiudere il gripper nel suo punto medio per favorirne una presa piú solida. Ad esempio, se l'oggetto é alto 10 cm, il gripper verrà chiuso ad altezza 5 cm. L'UR5, poi, si muoverá nell'altra posizione pubblicata sul topic in attesa di cominciare l'ultima fase dell'applicazione.

### 4.2.3 Inserimento

Supponendo di voler inserire l'oggetto in un contenitore avente, al centro, una cavità di uguale forma e di dimensioni simili, non sarà sufficiente utilizzare la posizione acquisita grossolanamente in 4.2.1, in quanto troppo poco precisa. A tal proposito in [17] viene mostrato il codice per calcolare il centro del contenitore (corrispondente al centro della cavità) in cui inserire l'oggetto. Utilizzando il controllore di velocità il braccio viene fatto muovere finché non viene rilevato un contatto con tutti i bordi. Vengono salvate le posizioni di ogni estremitá e vengono, poi, utilizzate per calcolare il punto esatto in cui inserire l'oggetto. Il robot potrà, quindi, posizionarsi al di sopra di tali coordinate e cominciare un processo di discesa che lo porterá ad incastrare l'oggetto precisamente nell'alloggio finale.

## 4.3 Movimento a spirale

Una possibile variazione del pick and place consiste nel far si che il braccio **trovi** la cavità in cui inserire l'oggetto. In 4.2.3 si presupponeva che la cavità si trovasse al centro del contenitore, in questo modo era possibile, determinandone il centro,

inserire precisamente l'oggetto nella posizione corretta. Ovviamente se il foro non si trova al centro, l'inserimento non andrà a buon fine. Per risolvere questo problema si é pensato di sostituire la parte di inserimento precedentemente descritta con un nuovo nodo in grado di trovare la posizione del foro [18]. Quando il braccio si trova in contatto con il contenitore, comincia ad effettuare un movimento a **spirale**. Mentre effettua tale movimento, mantiene l'oggetto in contatto con la superficie del contenitore. Se il sensore non rileva piú alcuna forza lungo l'asse  $z$ , significa che ci si trova in uno dei seguenti casi:

- il contenitore non ha una superficie piana. Il foro non é ancora stato trovato e quindi é necessario far scendere il braccio per ristabilire il contatto e continuare a cercare.
- il braccio si trova al di sopra del foro. Si può procedere con l'inserimento dell'oggetto.

Per implementare questa funzionalità si é pensato di far scendere il braccio ogni qual volta il sensore non rileva piú una forza lungo l'asse  $z$ . Se la differenza di altezza é superiore ad una determinata soglia, significa che probabilmente si é trovato il foro e quindi il gripper si aprirá per favorire l'inserimento dell'oggetto. A differenza della versione mostrata in 4.2, questa non raggiunge sempre l'obiettivo. Può capitare, infatti, che il braccio non trovi mai la cavità per via dell'incremento del raggio della spirale e che finisca al di lá dei bordi della scatola. Inoltre, se non si trova perfettamente al di sopra del foro, potrebbe non cominciare la fase di discesa non portando a termine il compito.

## 4.4 Trasporto collaborativo

Modificando ulteriormente l'inseguitore di forza, come mostrato in [19], si può implementare un'applicazione per il trasposto collaborativo uomo-robot. Rispetto alla versione utilizzata nella Sezione 4.1 é stata aggiunta la lettura delle torsioni misurate dal sensore. Ad esempio, nel pick and place, le torsioni lungo l'asse  $z$  venivano interpretate come l'input da parte dell'operatore per il salvataggio delle posizioni. In questo caso, invece, quando il sensore misura una torsione, essa viene interpretata come la volontà dell'operatore di ruotare il pezzo che viene trasportato. Allo stesso modo delle forze, viene calcolata la **velocità angolare** con cui far ruotare l'end effector dell'UR5, in modo tale da riuscire a seguire tutte le intenzioni dell'utente. La possibilità di rotazione dell'end effector porta, però,

ad un problema con i sistemi di riferimento. Infatti, `twist_controller`, effettua i movimenti rispetto al sistema di riferimento dell'end effector, ma se esso viene ruotato sarà necessario cambiare il verso di movimento per seguire correttamente le forze in input. Per ovviare a questo problema, sono state utilizzate le **trasformazioni geometriche** da un sistema di riferimento ad un altro. Utilizzando la libreria `tf`, è possibile convertire le coordinate di un punto in un sistema di riferimento, nelle coordinate di un altro sistema di riferimento connesso ad esso. In questo modo è stato possibile inviare i comandi `Twist` al robot, rispetto ad un sistema di riferimento fisso (quale la base del robot), in modo tale che il vettore velocità calcolato non fosse dipendente dall'orientazione dell'end effector.



## **Capitolo 5**

## **Conclusioni**





# Bibliografia

- [1] *Quigley, Morgan, et al. "ROS: an open-source Robot Operating System."*  
*ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.*
- [2] *ROS Tutorials*, <http://wiki.ros.org/ROS/Tutorials>.
- [3] *Catkin Workspace*, <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>.
- [4] *You Tube Tutorial*, <https://www.youtube.com/playlist?list=PLLSegLrePWgIbIrA4iehUQ-impvIXdd9Q>.
- [5] *Esempio topic e service*, <https://github.com/andreastocco01/ros/tree/main>.
- [6] *Creating ROS msg and srv* <http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv>.
- [7] *Download material for UR5* <https://www.universal-robots.com/cb3/>.
- [8] *Download FT300-S manual* <https://robotiq.com/support/ft-300-force-torque-sensor>.
- [9] *FT300-S driver* [https://github.com/andreastocco01/ft300\\_driver](https://github.com/andreastocco01/ft300_driver).
- [10] *Robotiq maintained repo* <https://github.com/TAMS-Group/robotiq>.
- [11] *MoveIt Tutorial* [https://ros-planning.github.io/moveit\\_tutorials/](https://ros-planning.github.io/moveit_tutorials/).
- [12] *Ambiente di lavoro* [https://github.com/andreastocco01/environment\\_setup](https://github.com/andreastocco01/environment_setup).
- [13] *Controller Manager* [http://wiki.ros.org/controller\\_manager](http://wiki.ros.org/controller_manager).
- [14] *Viscosity* [https://github.com/andreastocco01/ur5\\_ft\\_tasks/blob/main/src/viscosity.cpp](https://github.com/andreastocco01/ur5_ft_tasks/blob/main/src/viscosity.cpp).

- [15] *Force Follower* [https://github.com/andreastocco01/ur5\\_ft\\_tasks/blob/main/src/velocity\\_force\\_follower.cpp](https://github.com/andreastocco01/ur5_ft_tasks/blob/main/src/velocity_force_follower.cpp).
- [16] *Placement* [https://github.com/andreastocco01/ur5\\_ft\\_tasks/blob/main/src/placement.cpp](https://github.com/andreastocco01/ur5_ft_tasks/blob/main/src/placement.cpp).
- [17] *Insertion* [https://github.com/andreastocco01/ur5\\_ft\\_tasks/blob/main/scripts/place\\_ontop.py](https://github.com/andreastocco01/ur5_ft_tasks/blob/main/scripts/place_ontop.py).
- [18] *Spiral movement* [https://github.com/andreastocco01/ur5\\_ft\\_tasks/blob/main/src/spiral\\_movement.cpp](https://github.com/andreastocco01/ur5_ft_tasks/blob/main/src/spiral_movement.cpp).
- [19] *Full force follower* [https://github.com/andreastocco01/ur5\\_ft\\_tasks/blob/main/src/full\\_velocity\\_force\\_follower.cpp](https://github.com/andreastocco01/ur5_ft_tasks/blob/main/src/full_velocity_force_follower.cpp).