Università degli Studi di Padova

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 attivitá 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, dopo averne validato l'accuratezza attraverso alcuni esperimenti.

Indice

Introduzione

I robot manipolatori hanno rivoluzionato l'automazione industriale, permettendo lo svolgimento di operazioni complesse in modo rapido, preciso e sicuro. Uno dei modelli piú utilizzato é l'**UR5** di **Universal Robot**. Per via della sua flessibilitá ed efficienza si é scelto di utilizzarlo in questo studio. 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 attivitá. La loro versatilitá li rende strumenti preziosi in molti campi tra cui: la **robotica industriale** e la **medicina**. Essi infatti consentono al robot di controllare la forza esercitata durante le operazioni di assemblaggio, levigatura, saldatura e manipolazione degli oggetti. Vengono inoltre utilizzati nella riabilitazione, per valutare la forza muscolare e i progressi del paziente. 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** sfruttano la proprietá di alcuni materiali (cristalli piezoelettrici) di generare una carica elettrica se sottoposti a deformazione meccanica. Tale variazione puó essere misurata per determinare la forza o la coppia applicata. Il sensore **FT 300-S** di **Robotiq** sfrutta questo principio di funzionamento ed é in grado di misurare forze e coppie lungo i sei gradi di libertá (x, y, z, roll, pitch, yaw). É stato necessario installarlo manualmente sull'UR5 perché, a differenza di altri, non possiede sensori coppia-forza integrati. In questa tesi verrá 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 é ampiamente utilizzato dalla comunitá informatica perché fornisce strumenti e librerie per il controllo e la comunicazione tra le componenti di un sistema robotico. Alcuni test per la valutazione delle prestazioni del sensore in termini di reattivitá e precisione verranno descritti nel Capitolo 4. Nel Capitolo 5, invece, verranno presentate delle possibili applicazioni volte a dimostrare

l'efficacia di tali sensori per lo svolgimento di attivitá 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 possibilitá di registrare e riprodurre dati, favorendo cosí il debugging e il testing delle applicazioni. La versione raccomandata e utilizzata é **ROS Noetic** per **Ubuntu Focal 20.04**. Questo capitolo ci fornirá 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 `rosrun <nome_pacchetto> <nome_nodo>`. Tuttavia si incorrerá in un errore se prima non viene fatto partire un ROS Master. Lo scopo principale di un **ROS Master** é quello di consentire ai singoli nodi di localizzarsi a vicenda. Una volta fatto questo, essi potranno comunicare tra loro attraverso topic o service. Per far

partire un ROS Master sará 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**.

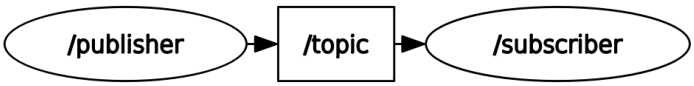


Figura 1.1: Schema di comunicazione

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 piú vecchi verranno cancellati per fare spazio a quelli piú 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 possibilitá 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 daremo il permesso di farlo. Ci sono due modi per fare ció:

- `ros::spinOnce()` chiede a ROS di eseguire tutte le callback in sospeso e poi ci restituisce il controllo
- `ros::spin()` chiede a ROS di attendere e di eseguire tutte le callback in sospeso 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 altro modo in cui i nodi possono comunicare tra loro. 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 potrá creare una richiesta nel formato specificato e inviarla al nodo service come parametro della funzione `call()`. Il nodo service elaborerá quindi la richiesta fornendo una risposta al client.

1.4 Topic e Service a confronto

I topic e i service sono due modalitá 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 piú adatti per la trasmissione di flussi di dati continui (come quelli provenienti dai sensori) e i service piú 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 modalitá 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 potrá 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.

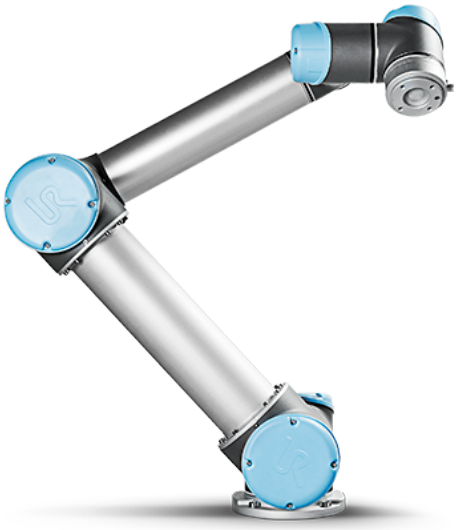


Figura 2.1: UR5/CB3

All'estremitá 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 gravitá, ossia la distanza tra l'estremitá del braccio robotico (punto di applicazione dell'end effector) e il centro di gravitá dell'UR5.

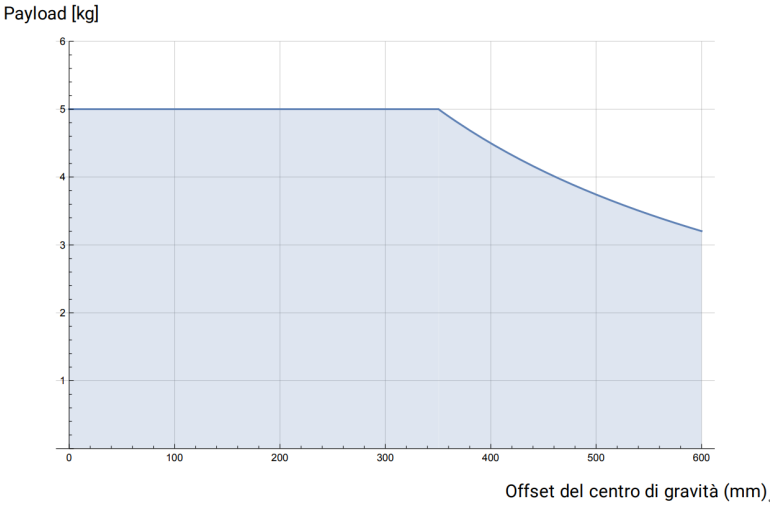
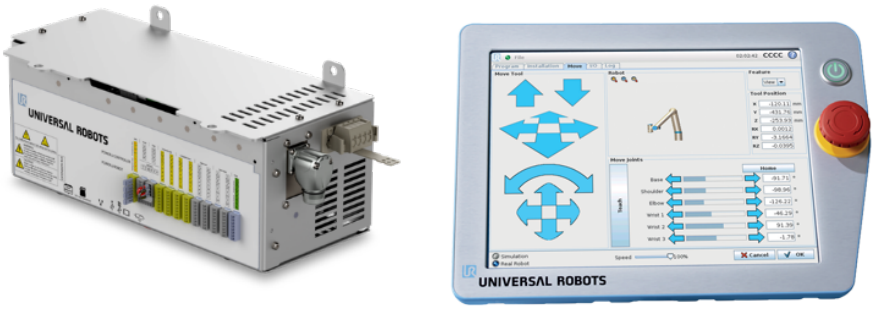


Figura 2.2: Andamento del carico massimo sopportabile rispetto all'offset del centro di gravitá

Come si puó 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 alla cosiddetta **control box** che é un dispositivo collegato alla rete elettrica 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 e il controllo remoto, a cui é stato collegato il computer 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 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 possibilitá di installazione di un end effector, che puó essere facilmente posizionato 'al di sopra' del sensore. L'FT300-S é in grado di rilevare forze e torsioni nel range di $\pm 300N$ e $\pm 30Nm$ rispettivamente. Le misurazioni del sensore hanno un rumore di fondo intrinseco, é quindi 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 modalitá 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 modalitá 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

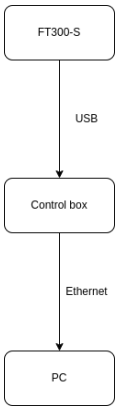


Figura 2.5: Schema collegamento

Con la configurazione mostrata in Figura 2.5, per poter leggere i dati provenienti dal sensore é stato necessario sviluppare un **driver**. Per prima cosa si é stabilita una connessione di rete tramite un **socket** collegato all'indirizzo IP del robot, in particolare alla porta **63351**. Su tale porta il sensore invierá 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 piú 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

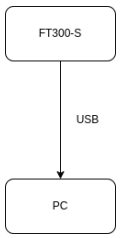


Figura 2.6: Schema collegamento

Con questa configurazione, invece, un driver per la lettura dei dati del sensore ci viene giá fornito da Robotiq. Una volta scaricato il loro repository GitHub [10], per eseguire il driver é sufficiente far partire un nuovo nodo ROS con il comando `rosrun robotiq_ft_sensor rq_sensor`. Il driver consente l'utilizzo di entrambe le modalitá 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, é 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, 4 si é 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 parlerá 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 ulteriormente, é peró 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. Per semplificare il processo di configurazione e setup del sistema, MoveIt mette a disposizione il **MoveIt Setup Assistant**. Si tratta di uno strumento che fornisce un'interfaccia grafica per consentire agli utenti di personalizzare i file di configurazione che verranno successivamente generati da questo strumento.

Capitolo 3

Validazione del sensore

Capitolo 4

Applicazioni industriali

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] *YouTube 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.

[10] *Robotiq maintained repo* https://github.com/TAMS-Group/robotiq.