Università degli Studi di Padova

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di Laurea Triennale
in Ingegneria Meccatronica

# Pick and Place tramite ROS2 e MoveIt2 per robot Franka Emika Panda

*Relatore:*
Ch.mo Prof. Stefano Michieletto

*Laureando:*
Pietro Franceschini
1195790

Anno Accademico 2021/2022

# Sommario

Il lavoro di tesi consiste nella programmazione del robot Franka Emika Panda e nell'implementazione di un setup per l'esecuzione di operazioni pick and place. Queste hanno per oggetto una serie di cubi di piccole dimensioni, identificabili tramite dei tag posizionati superiormente ad essi, la cui posizione è individuata da una telecamera esterna che comunica con il robot. La prima parte della tesi si concentra su una simulazione del setup in ambiente virtuale attraverso l'utilizzo dei software ROS2 e MoveIt2. La seconda descrive gli esperimenti di laboratorio eseguiti con il robot sulla base del codice precedentemente sviluppato in fase di simulazione.

# Introduzione

Il seguente lavoro tratta la movimentazione di un braccio robotico all'interno di un ambiente virtuale e successivamente in uno reale. Nei seguenti capitoli vengono approfonditi i software base di controllo, la loro intercomunicazione e le differenze tra i vari processi attuati. Particolare attenzione è messa nella scrittura del codice di controllo in linguaggio C++ per il funzionamento del robot collaborativo Panda di produzione della Franka Emika, comprensivo del manipolatore Panda Hand, con lo scopo di effettuare l'azione fondamentale di Pick & Place. Ogni software utilizzato viene descritto in modo rapido ma dettagliato, soffermandosi sulle particolarità e funzionalità, con attenzione all'architettura e alle caratteristiche principali. Vengono quindi citati i software per programmazione e movimentazione ROS2, MoveIt2, oltre a quello per la visualizzazione Rviz. Una parentesi importante riguarda la descrizione e le applicazioni dei robot collaborativi e del cobot preso in studio, inoltre vengono approfondite le caratteristiche che li contraddistinguono, con una rapida comparazione tra robot industriali e una sezione riguardante le norme vigenti e gli utilizzi futuri. Si conclude discutendo di eventuali sviluppi futuri come una metodologia di interfacciamento tra ambiente reale e virtuale effettuata attraverso AprilTag e fotocamera Kinect.

# Indice

# Elenco delle figure

# Capitolo 1

# Robot Collaborativi

## 1.1 Cobot

I Cobot, o robot collaborativi, hanno avuto negli ultimi anni una diffusione esponenziale dovuta principalmente a differenti fattori come versatilità, facilità di programmazione, velocità e sicurezza.
Ma cosa sono i cobot?
Vengono definiti come robot antropomorfi con movimenti su almeno sei assi, progettati per rispettare criteri di sicurezza, flessibilità, compattezza e studiati per lavorare a stretto contatto con l'operatore anche senza barriere protettive all'intorno. I primi sono stati introdotti da aziende come la General Motors, Cobotics, KUKA, FANUC e Universal Robot.
Si prende quest'ultima come esempio di classificazione dei modelli, identificati dalla sigla UR (Universal Robot) e un numero per indicare la portata massima e il raggio d'azione:

| Denominazione | Portata (Kg) | Raggio d'azione (mm) |
|---|---|---|
| UR3 | 3 | 500 |
| UR5 | 5 | 850 |
| UR10 | 10 | 1300 |
| UR16 | 16 | 900 |

Previa analisi della sicurezza dell'ambiente di lavoro, come citato precedentemente, i cobot possono essere utilizzati senza l'ausilio di barriere di protezione grazie all'implementazione di funzioni come tempo di arresto e distanza personalizzabili, ottenendo una condivisione di spazio di lavoro proibitiva nel caso di robot industriali.

**Quando un robot può dirsi collaborativo e quali sono le principali differenze da un robot industriale?**
Quello che fa di un robot un cobot è la compattezza e sicurezza, unita ad una facilità di programmazione e di utilizzo oltre che ad una agevole integrazione in eventuali catene di montaggio.
Le differenze dai robot industriali sono le dimensioni, il costo e la metodologia di programmazione ma soprattutto, la sicurezza, la flessibilità e la velocità di messa in opera.

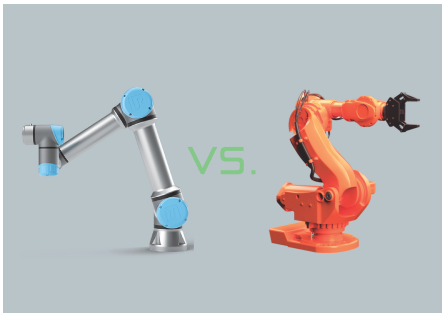


**Figura 1.1:** Sulla sinistra un cobot, sulla destra un robot industriale

**Sicurezza:**La maggior parte può operare senza l'ausilio di barriere di protezione, a differenza dei robot tradizionali che necessitano di costose recinzioni per garantire un'adeguata separazione tra ambiente di lavoro operatore-macchina.
In un cobot invece è possibile impostare distanza e tempo di arresto e molte altre variabili che rendono la sua implementazione sicura anche in un'area affollata, inoltre sono equipaggiati con un'ampia gamma di sensori che ne accrescono la sicurezza: laser scanner, accelerometri e sistemi di visione che registrano la presenza dell'operatore controllando movimenti e tempi di reazione.

- **Stop di sicurezza monitorato:** il sistema si arresta prima che i lavoratori possano accedere o essere esposti a qualsiasi situazione di pericolo.

- **Guida manuale:** l'operatore utilizza un dispositivo manuale o muove fisicamente il braccio del robot per aiutarlo a completare la sua operazione.

- **Monitoraggio della distanza e della velocità:** il dispositivo di sicurezza esterno fa sì che il robot rallenti se i requisiti minimi di distanza con l'operatore non sono rispettati.

- **Limitazione della potenza e della forza:** arresto istantaneo per evitare lesioni alle persone quando il livello massimo di forza o di potenza viene raggiunto.

**Flessibilità:** Un robot industriale tradizionale risulta performante solo su grandi volumi produttivi. L'automazione cui un robot tradizionale dà origine è rigida. Un cobot invece è flessibile perché piccolo e leggero: è quindi agevole lo spostamento all'interno del layout industriale anche grazie alla facilità e velocità di messa in opera.

**Velocità di messa in opera:** Non richiedono lunghe e costose modifiche al layout produttivo per quanto già citato, e possono essere integrati in tempi rapidi nelle aree di lavoro. Vengono alimentati con una tensione di 220V, quindi utilizzabili anche fuori da ambienti industriali.

## 1.2 Ambiti di utilizzo

I cobot sono in grado di automatizzare un gran numero di applicazioni, ne vengono elencate alcune:

- Pick&Place;
- Asservimento macchine;
- Manipolazione materiale;
- Controllo qualità;
- Assemblaggio;
- Finitura.

**Figura 1.2:** Esempi di lavorazione mediante cobot

Si decide di soffermarsi maggiormente sulla funzione Pick&Place, oggetto di studio della seguente tesi. Consiste in un ciclo operativo durante il quale i prodotti

vengono prelevati da un'area specifica dell'ambiente di lavoro e organizzati in un'altra, risparmiando operazioni ripetitive all'operatore umano.
Per via della movimentazione su sei assi e della possibilità di integrare numerosi accessori, permettono di gestire oggetti di diverse forme, dimensioni e materiali, entro i limiti di portata e di raggio di lavoro, con precisione e sicurezza.
Nel corso degli anni i cobot hanno avuto una importantissima evoluzione tanto da essere considerati indispensabili per la maggior parte delle aziende anche di piccole dimensioni, o in ambiti precedentemente non considerati come logistica o alimentare, grazie alla possibilità di ripetizione dello stesso movimento innumerevoli volte e per lunghi periodi di tempo con precisione invidiabile.
Il loro grado di ripetibilità dei movimenti è imparagonabile a quello umano: riescono ad eseguire operazioni non ergonomiche, evitando agli operatori rischi per la salute associati a posture forzate, movimenti ripetitivi o movimentazione manuale dei carichi. Per le operazioni a basso valore aggiunto che non richiedono l'intervento umano diretto o in operazioni in ambiente rischioso, i cobot sono lo strumento ideale. Essi manterranno il loro grado di precisione e accuratezza, lasciando altre funzioni più specifiche e di alto livello ai lavoratori.

## 1.3 Regolamentazioni

La norma *EN ISO 10218*, unitamente alla Direttiva Macchine, e la specifica *ISO/-TS 15066* hanno definito i requisiti di sicurezza per i robot collaborativi. In tale contesto, la definizione di robot collaborativo comprende sia gli strumenti adattati sul braccio per eseguire determinati compiti che gli oggetti da quest'ultimo movimentati.
Una cooperazione ravvicinata o il contatto diretto tra l'operatore e il robot può comportare l'insorgere del rischio di collisione, di conseguenza, la valutazione dello stesso deve anche comprendere il luogo di lavoro previsto.[1]

## 1.4 Costruzione

Come detto in precedenza i cobot, per essere definiti tali, devono possedere almeno 6 gradi di libertà: tre di traslazione (x,y,z) e tre di rotazione (rollio, beccheggio, imbardata). Fig. 1.3
Per ogni braccio sono previste apposite giunture, soprattutto giunti rotoidali che permettono anche rotazioni complesse.

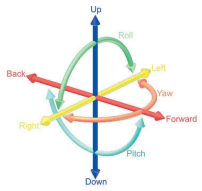


**Figura 1.3:** I 6 gradi di libertà

Costituito da una base fissa all'interno della quale vengono situati i connettori per il controllo e la gestione dati, prosegue con gli altri membri detti: colonna, braccio, avambraccio, corpo del polso, flangia e testa. La testa è la parte meccanica costituita dallo strumento od oggetto adibito alla lavorazione, come una pinza robotica nel caso preso in tesi.

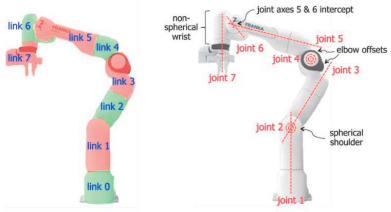


**Figura 1.4:** Suddivisione del cobot in link e giunti

I connettori all'interno della base, posizionati anche internamente al corpo del polso, vengono collegati tramite appositi fili all'Unità di Controllo, spesso anche un comune PC, che monitora e gestisce il movimento degli assi.
L'Unità di Controllo si interfaccia sia con il "braccio" che con il Terminale di Programmazione da cui vengono inviate le istruzioni. [2]

## 1.5 Futuro dei Cobot

L'enorme interesse esercitato dalle imprese di diversi settori ha permesso una crescita esponenziale del mercato dei cobot. La ricerca sul miglioramento delle prestazioni, della sicurezza, e dei software di controllo è continua, data la grande sensibilizzazione riguardo la sostenibilità ambientale per ridurre sprechi, consumo di risorse e scarti di produzione; tutto a vantaggio di una maggiore efficienza e di un minor impatto ambientale. I cobot infatti lavorano con maggior costanza, qualità, ripetibilità, assicurando standard di prodotto uniformi, riducendo i pezzi di scarto e centrando l'obiettivo di una produzione a 0 difetti.
Viene stimato che i processi produttivi in modalità collaborativa siano più efficienti fino all'85% portando alle aziende migliori performance economiche. Si

pensa che in un futuro non troppo lontano i cobot possano far parte della vita casalinga di tutti. [3] [4]

# Capitolo 2

# Ambienti di sviluppo

## 2.1 Hardware

### 2.1.1 Franka Emika Panda

Franka Emika Panda è un robot di tipo collaborativo ad alto livello prestazionale ed elevata integrazione meccatronica. Facilmente implementabile in ambito industriale o in ambiente pubblico oltre che di semplice inserimento in sistemi multi-robot.
Vengono riportate di seguito alcune delle sue funzioni:

- Soft / high accuracy / heavy pick and place;
- Assemblaggio;
- Testing del graphic user interface (GUI) anche in modalità soft;
- Apprendimento delle funzioni manuale o tramite applicazione web.

Il sistema robotico Panda della Franka Emika è composto da un braccio (Arm), un controllore e una locazione per un utensile. Il braccio possiede 7 gradi di libertà con sensori di coppia in ogni giunto, permettendo un'avanzata gestione della coppia e una rigidità/elasticità regolabile.
Dal datasheet riportato in figura 2.4 si possono evidenziare alcune caratteristiche come il carico utile di 3 Kg, il raggio d'azione di 855 mm e la presenza di limiti degli angoli dei giunti.
La figura 2.1 mostra una tipica zona di lavoro ed evidenzia che il robot è fornito di alcuni dispositivi di sicurezza per abilitazione, azionamento e stop dalla distanza. Inoltre possiede un sistema di sicurezza che blocca i 7 giunti non appena viene

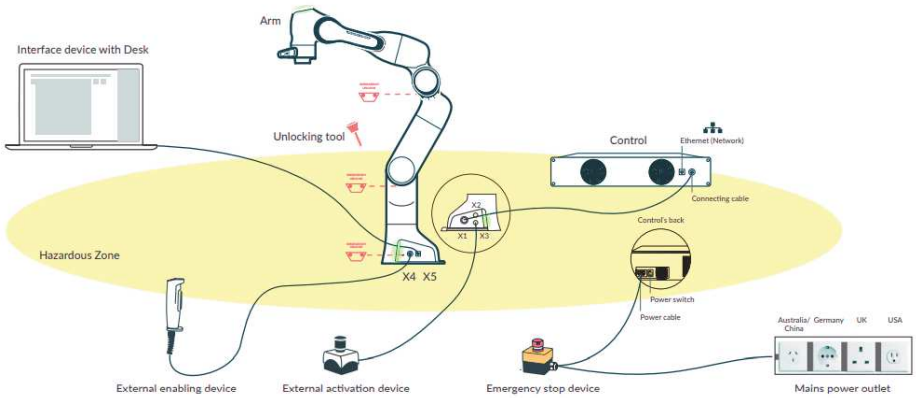


**Figura 2.1:** Panda nella zona di lavoro

spenta l'alimentazione o in caso di urto, mantenendo il braccio in posizione. In caso di emergenza è comunque possibile sbloccare manualmente il sistema, inserendo una specifica chiave in uno dei tre fori disposti lungo il braccio.
Sulla base del manipolatore e sopra l'end effector sono presenti LED per indicare lo stato di funzionamento secondo un codice di colori riportato nella figura 2.2 sottostante.

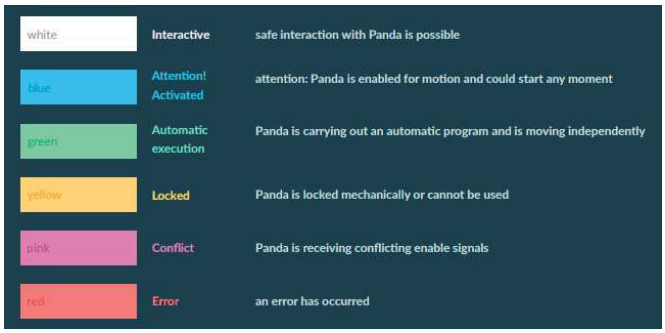


**Figura 2.2:** Codice colori

È presente inoltre un'interfaccia grafica a basso livello chiamata FCI (Franka Control Interface), utilizzabile solo in ambito di ricerca, per ottenere una connessione bidirezionale veloce al robot. Ideale per condurre ricerche e test su algoritmi di pianificazione del movimento, strategie di gripping e algoritmi di machine learning.

### Potenzialità e funzionalità

Vengono di seguito riportate le informazioni del manuale d'istruzione per avere una visione generale delle sue potenzialità.
[5]

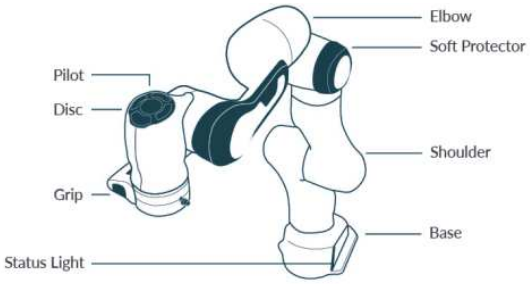


**Figura 2.3:** Nomenclatura link Panda

- Soft Robot performance:
  - Movimento: Ripetibilità di posizionamento di ± 0.1mm e una deviazione di percorso trascurabile anche a velocità superiori a 2m/s permettendo precisione, robustezza e velocità di esecuzione dei processi richiesti.
  - Sensore di forza:
    * Sensori di coppia e di forza in ogni giunto dei 7 assi;
    * Algoritmi di controllo con tempi di reazione in millisecondi;
    * Controllo di coppia e forza flessibile per adattamento al compito assegnato o all'ambiente;
  - Controllo di forza di 1kHz: forza minima attuabile pari a 0,05N per lavorazioni come il soft pick and place, tracciatura contorni o lucidatura;
  - Interazione: modalità di manipolazione regolabili per compensazione della gravità e riduzione dell'attrito del 60% garantendo interazione fisica fluida e semplice con l'utente;
  - Impedenza: capacità di maneggiare oggetti fragili e di piccole dimensioni senza danneggiarli.
- Tipologia di movimentazione:
  - Joint space: sette giunti in movimentazione simultanea dalla posizione iniziale di partenza ad una definita di arrivo. L'end effector non esegue una rotazione specifica ma la sua posizione è definita dal movimento degli altri giunti;

  - Cartesian space: i sette giunti si muovono al solo fine di portare l'end effector nella giusta posizione. I movimenti nello spazio cartesiano permettono di tracciare percorsi precisi e ben definiti come linee rette;
  - Redundancy: il braccio può raggiungere la posizione richiesta attraverso molteplici traiettorie dei giunti, così da effettuare la migliore al fine di evitare collisioni e diminuire tempi di trasferimento.

- Sensibiità: ogni giunto possiede un sensore di coppia reale per riconoscere e reagire in caso di contatto, anche di piccola entità, aumentando di molto la sicurezza intrinseca e consentendo l'esecuzione di tragitti guidati dall'esterno (tool path recognition)

  - Collision detection and reaction: i sensori forniscono i dati di coppia reali applicati a ciascun giunto e, in combinazione con il modello di controllo, si può rilevare la differenza tra coppia prevista e quella effettiva facendo agire il braccio di conseguenza. Se la differenza tra coppia attesa e reale è al di sopra di una certa soglia, viene rilevata una collisione fermando istantaneamente l'esecuzione;
  - Generazione di forza:Possibilità di modulare la forza in caso sia necessario sbloccare il braccio perché in contatto con un oggetto che ne impedisce il movimento.

**Ecosistema**

Franka Emika, attraverso ricerche interne e collaborazioni con altre aziende, ha sviluppato numerose estensioni per rendere il sistema robotico integrabile e immediatamente produttivo in vari scenari industriali. Oltre al predefinito gripper robotico a pinza "Franka Hand", ne è stato prodotto uno basato su pompe a vuoto elettriche in grado di movimentare oggetti sottili e fragili come schermi smartphone. [6]

# PANDA - DATASHEET

May 2019

## HARDWARE

### Arm

| | |
|---|---|
| Degrees of freedom | 7 |
| Payload | 3 kg |
| Workspace | see backside |
| Maximum reach | 855 mm |
| F/T Sensing | link-side torque sensors in all 7 axes |
| Expected nominal lifetime [3,4] | 20,000 h |
| Joint position limits [°] | A1, A3, A5, A7: -166/166<br>A2: -101/101<br>A4: -176/-4<br>A6: -1/215 |
| Mounting flange | DIN ISO 9409-1-A50 |
| Installation position | upright |
| Weight | ~ 17.8 kg |
| Moving mass | ~ 12.8 kg |
| Protection rating | IP30 |
| Ambient temperature [2] | 15 – 25 °C (typical)<br>5 – 45 °C (extended) |
| Air humidity | 20 – 80 % non-condensing |
| Power consumption | • max. ~ 350 W<br>• typical application ~ 60 W |
| Interfaces | • ethernet (TCP/IP) for visual intuitive programming with Desk<br>• input for external enabling device<br>• input for external activation device or safeguard<br>• Control connector<br>• Connector for end-of-arm tooling |

### Control

| | |
|---|---|
| Controller size (19") | 355 x 483 x 89 mm (D x W x H) |
| Supply voltage | 100 – 240 $V_{AC}$ |
| Mains frequency | 47 – 63 Hz |
| Power consumption | ~ 80 W |
| Active power factor correction (PFC) | yes |
| Weight | ~ 7 kg |
| Protection rating | IP20 |
| Ambient temperature | 15 – 25 °C (typical)<br>5 – 45 °C (extended) |
| Air humidity | 20 – 80 % non-condensing |
| Interfaces | • ethernet (TCP/IP) for internet and/or shop-floor connection<br>• power connector IEC 60320-C14 (V-Lock)<br>• Arm connector |



## SOFT-ROBOT PERFORMANCE

### Motion

| | |
|---|---|
| Joint velocity limits [°/s] | A1, A2, A3, A4: 150<br>A5, A6, A7: 180 |
| Cartesian velocity limits | up to 2 m/s end effector speed |
| Pose repeatability | +/- 0.1 mm (ISO 9283) |
| Path deviation [3] | +/- 1.25 mm |

### Force

Sensing [3]

| | |
|---|---|
| Force resolution | <0.05 N |
| Relative force accuracy | 0.8 N |
| Force repeatability | <0.15 N |
| Force noise (RMS) | <0.035 N |
| Torque resolution | <0.02 Nm |
| Relative torque accuracy | 0.15 Nm |
| Torque repeatability | <0.05 Nm |
| Torque noise (RMS) | <0.005 Nm |

1 kHz Control [3]

| | | |
|---|---|---|
| Minimum controllable force (Fz) | | 0.05 N |
| Force controller bandwidth (-3 dB) | | 10 Hz |
| Force range [N] | Nominal case | Best case |
| Fx | -125 – 95 | -150 – 115 |
| Fy | -100 – 100 | -275 – 275 |
| Fz | -50 – 150 | -115 – 155 |
| Torque range [Nm] | Nominal case | Best case |
| Mx | -10 – 10 | -70 – 70 |
| My | -10 – 10 | -16 – 12 |
| Mz | -10 – 10 | -12 – 12 |

### Interaction

| | |
|---|---|
| Guiding force | ~ 2 N |
| Collision detection time | <2 ms |
| Nominal collision reaction time [3,4] | <50 ms |
| Worst case collision reaction time [3] | <100 ms |
| Adjustable translational stiffness | 0 – 3000 N/m |
| Adjustable rotational stiffness | 0 – 300 Nm/rad |
| Monitored signals | Joint position, velocity, torque<br>Cartesian position, velocity, force |

## ADD-ONS

| | |
|---|---|
| Safety retrofit option with safety-rated PLC | PLd Cat. 3<br>• Safe torque off (STO)<br>• Safe OSSD inputs |
| Fully integrated end effectors | • 2-finger gripper<br>• Vacuum gripper |
| Fast mounting | Paw |
| Demonstration | Pop-up Box |
| Research interface | 1kHz Franka Control Interface |
| Fieldbuses | Modbus/TCP, OPC UA, Profinet |

**Figura 2.4:** Datasheet Franka Emika Panda [7]

[7]

### 2.1.2 Panda Hand

Il manipolatore principale della Franka Emika integrabile con Panda cobot è denominato Franka Hand, ovvero una pinza robotica a *finger* paralleli, le cui caratteristiche tecniche vengono presentate nella tabella seguente:

| | |
|---|---|
| Larghezza massima | 80 mm |
| Velocità massima | 50 mm/s |
| Forza di picco | 140 N |
| Forza statica | 70 N |

Comunica per connessione diretta al braccio ed è alimentata dallo stesso senza necessità di un cablaggio esterno. [8]

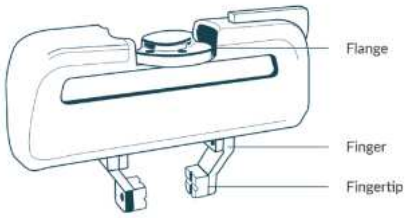


**Figura 2.5:** Franka Hand

### 2.1.3 Azure Kinect

Azure Kinect DK è un dispositivo compatto all-in-one con più modalità di funzionamento per la mappatura dell'ambiente circostante. Al suo interno sono presenti numerosi sensori come microfoni spaziali, sensori di profondità, sensori di orientamento, videocamere di cui una RGB. Una volta interfacciato con il PC sarà possibile utilizzarlo per scansionare oggetti in 3D, per riconoscimento facciale o ancora per traduzione istantanea, sintesi e riconoscimento vocale. [9]

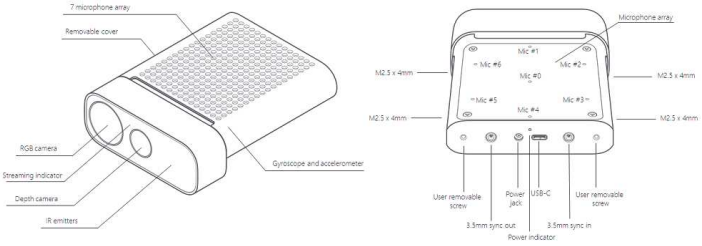


**Figura 2.6:** Azure Kinect

## 2.2 Software

### 2.2.1 ROS2

**Introduzione**

La progettazione di un sistema robotico richiede un'elevata capacità computazionale, dovendo gestire numerosi software come quelli per la visualizzazione del sistema virtuale in fase di simulazione, un IDE per lo sviluppo del codice o altri strumenti utili al programmatore per creare, aggiornare ed effettuare operazioni di debugging. Recentemente però, l'espansione tecnologica ha aperto la strada al controllo mediante semplici personal computer. Tutto ciò è stato accompagnato dall'avvento di ROS, un software open-source per lo sviluppo e la programmazione di robot, realizzato inizialmente dalla Stanford AI Laboratory nel 2007. [10]
ROS offre una piattaforma software standardizzata utile in ambito di ricerca, prototipazione, creazione, implementazione e produzione finale in cui il programmatore sviluppa il proprio progetto. Fornisce inoltre dei servizi base come il controllo di dispositivi tramite driver, la comunicazione tra processi, un build system, la gestione delle applicazioni (packages) e l'astrazione dell'hardware essenziale per l'esecuzione di software su hardware non specificatamente concepiti per richieste così complesse.
In generale ROS aiuta il programmatore a sviluppare il codice per l'esecuzione di uno specifico lavoro richiesto, ha inoltre un proprio compilatore che traduce il codice sorgente in linguaggio macchina. I linguaggi principali utilizzabili sono il C++ e il Python, mentre quelli per lo sviluppo delle librerie sono indipendenti dal linguaggio di programmazione utilizzato.
I processi avviati all'interno di ROS sono debolmente accoppiati e comunicano in rete in modalità peer-to-peer. Vengono rappresentati come una rete di nodi atti a ricevere ed inviare informazioni da e verso altri nodi (sensori e attuatori), ognuno dei quali è responsabile di una singola richiesta. [11] [12]
I nodi comunicano tramite:

- Pubblicazione e sottoscrizione di messaggi mediante Topic;
- Richiesta e risposta attraverso Services;
- Richiesta, feedback e risposta tramite Actions.

ROS è un sistema prevalentemente asincrono ma con la possibilità di integrare dei moduli per il funzionamento real-time, garantendo così l'esecuzione di una

richiesta entro una determinata scadenza temporale detta *deadline*.
L'estrema versatilità verso tutte le tipologie di robot, la possibilità di sviluppare il proprio codice senza licenza open-source, di visualizzare il proprio progetto in ambiente virtuale prima dell'implementazione in uno reale, rendono ROS lo strumento perfetto per l'utilizzo in numerose aziende e industrie che possiedono macchinari o processi automatizzati. [13]
ROS possiede una community molto vasta che ne permette continui aggiornamenti per la risoluzione di problemi e lo sviluppo di nuovi pacchetti ma ha anche permesso la creazione di una versione del software ancora più sicura e prestazionale chiamata ROS2, di cui si parla successivamente. [14]

### Architettura

ROS 2 è un framework completamente riprogettato sulla base del predecessore, per ovviare alle carenze presenti nell'ambito della sicurezza e nelle performance generali. Contiene nuovi pacchetti che possono essere installati facilitando la migrazione dai precedenti progetti scritti in ROS1 per sfruttare le nuove funzionalità e aggiornamenti API (Application Programming Interface) continui.
L'architettura di ROS 2 si basa su un sistema di comunicazione distribuito ed in tempo reale. Può essere visualizzata come suddivisione in più layer per una migliore comprensione, come in figura 2.7:

- Operating System layer: sistema operativo su cui tutti i processi sono avviati;
- DDS Implementation layer: si occupa dei protocolli di comunicazione;
- Abstract DDS layer: nasconde i dettagli dell'implementazione DDS;
- ROS 2 Client layer: contenente librerie fondamentali per il funzionamento del sistema;
- Application layer: codice utente con cui si definiscono i nodi dell'architettura.

[15]
**DDS (Data Distribution Service)**
ROS2 utilizza il middleware DDS, ovvero *servizio di distribuzione dati*, un intermediario per la comunicazione fra diversi processi a dispetto delle diversità dei

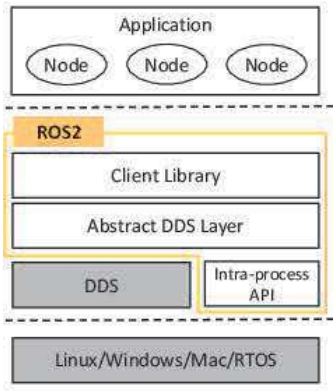


**Figura 2.7:** Architettura di ROS2 [16]

formati dei messaggi.
L'utilizzo di un DDS in ROS2, porta numerosi benefici in termini di scalabilità, affidabilità, sicurezza e flessibilità permettendo a tutti i nodi di comunicare tra loro all'interno dell'ambiente di distribuzione, senza la necessità di un nodo master che gestisca le comunicazioni come in ROS1.
Il livello di Ros Middleware Interface (rwm), al di sopra del DDS, si occupa dell'interfacciamento risparmiando al programmatore questa complessa procedura, invece di usare direttamente il middleware DDS. L'utente ha il permesso di selezionare e utilizzare la libreria DDS a sua scelta e grazie all'interoperabilità a livello di rete, è possibile sfruttare più librerie per lo stesso progetto. [16]

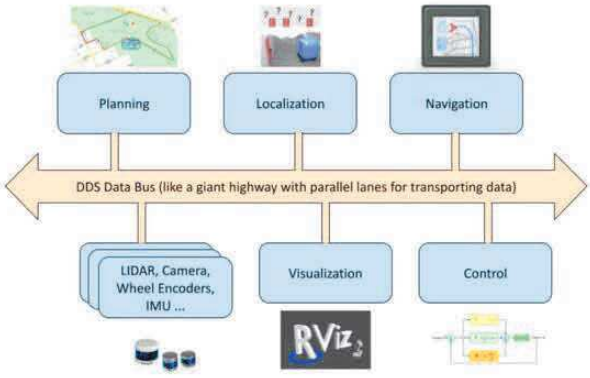


**Figura 2.8:** DDS: Data Distribution Service

### ROS2 Client Library

Qualsiasi applicazione di ROS 2 si interfaccia con le ROS Client Library (RCL). Tali RCL sono scritte attraverso il linguaggio di programmazione C e ognuna presenta un "RCLCPP client library" per linguaggio C++ e "RCLPY" per linguaggio

Python. Ne sono prodotte alcune anche con linguaggi indipendenti cosicché gli utenti possano utilizzare quello più adatto a loro e al progetto.
Per ROS client library si intendono librerie predefinite che permettono di implementare il codice ROS, introducendo logiche e funzionalità intrinseche come la scrittura di nodi, pubblicazione e sottoscrizione di topic e chiamate di servizi.
Vengono principalmente fornite con uno standard di interfaccia richiesta per lo scambio di dati tramite Topic e Services. Permettono anche operazioni di astrazione del sistema operativo e fruizione di microstrutture pronte all'uso.

**ROS2 Node Graph**

I nodi, i topic, i servizi e i messaggi formano l'architettura di base di un sistema ROS2.

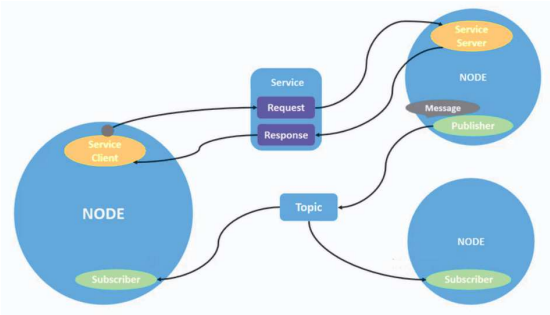


**Figura 2.9:** Trasferimento dei messaggi tra nodi

*Nodo:* è un eseguibile (generato da un insieme di codici sorgente), facente parte di un pacchetto. Ogni nodo rappresenta un modulo del sistema e svolge una determinata funzione. Ogni sistema robotico presenta molteplici nodi, i quali possono comunicare gli uni con gli altri attraverso i "Communication Patterns" inviando e ricevendo dati, sotto forma di messaggi, durante l'esecuzione di un processo. Ad esempio un nodo può occuparsi del controllo delle ruote motrici e un altro di un sensore laser o ancora, di eseguire una localizzazione. [17]
*Topic:* sono una modalità di comunicazione asincrona in broadcasting tra due nodi che identificano la tipologia di messaggio contenuto al loro interno. Un nodo pubblica un messaggio a cui viene assegnato un topic in base al contenuto, mentre un altro identifica il messaggio richiesto nel topic e lo sottoscrive, permettendo così la comunicazione. I topic possono essere visti anche come canali per lo scambio dei messaggi tra nodi, questi ultimi possono sia pubblicare che sottoscrivere simultaneamente diversi topic.
Sono utili soprattutto in caso di invio periodico di dati come quelli provenienti da sensori, o informazioni sullo stato dei robot in un sistema. [18]

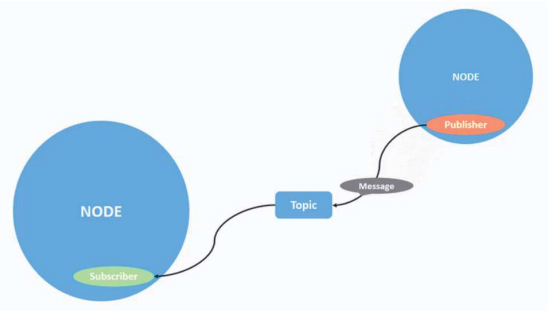


**Figura 2.10:** Comunicazione attraverso Topic

*Servizi:* sono una tipologia di comunicazione sincrona definita da una coppia di messaggi, uno per la richiesta e uno per la risposta. Un nodo server fornisce informazioni solo quando esse vengono richieste da un client (nodo richiedente). Si crea quindi un altro metodo di comunicazione tra nodi, meno flessibile del Topic, ma più funzionale in caso di necessità di una veloce comunicazione richiesta/risposta. [19]

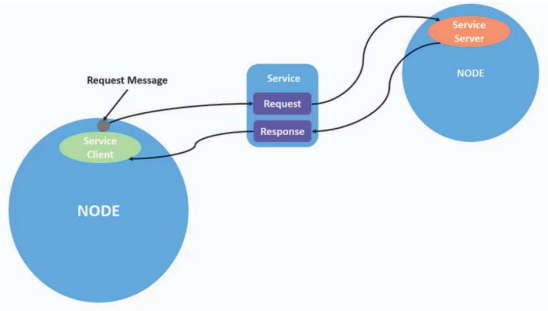


**Figura 2.11:** Comunicazione attraverso Service

*Azioni:* basate su un modello client-server e utilizzate per processi di lunga durata con un inizio e una fine ben definita. Composte da tre parti:

- Obiettivo;
- Feedback;
- Risultato.

Un nodo Client invia una richiesta con un obiettivo al nodo Server, ad esempio richiedendo il movimento di un robot verso una particolare zona dell'ambiente; successivamente il Server fornisce al Client un feedback costante del processo, come le posizioni frame per frame delle traiettorie; quindi il Server completa la richiesta e invia il risultato al Client.

Si nota una somiglianza con il sistema di comunicazione mediante Service, ma con una differenza sostanziale: le Action possono essere fermate in qualsiasi istante durante l'esecuzione. [20]

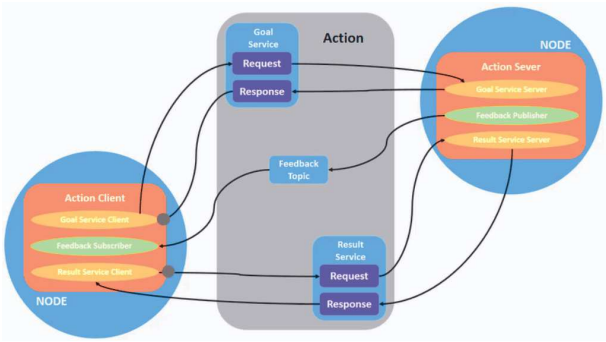


**Figura 2.12:** Comunicazione attraverso Action

*Messaggi:* sono le strutture dati trasmesse durante la comunicazione tra nodi. In queste strutture vengono accettati tipi primitivi standard (int, float, boolean) oltre a vettori di tipi primitivi ed eventualmente anche strutture nidificate di array. Tale struttura condivisa è definita in file dedicati (con estensione .msg) all'interno dei package che costituiscono il file system del progetto.
Tramite questo standard di comunicazione è possibile mettere in relazione diversi tipi di dispositivi sviluppati da differenti aziende, perfezionando l'integrazione nel sistema indipendentemente dal modello.

**Adattamento di dispositivi in ROS2**

I device, come sensori e attuatori, vengono integrati nel sistema ROS2 come nodi. Il codice richiesto per la programmazione è chiamato Adapter (wrapper). L'approccio è simile a quello di un sistema con "architettura modulare", ovvero un design in grado di generare moduli intrinsecamente interoperabili e collegabili senza necessità di riprogettazione. [21]

**Parameter Server**

Il Parameter Server è un database condiviso tra tutti i nodi del sistema e di facile accesso tramite API. E' caratterizzato da basse prestazioni perchè utilizzato per stipare e prelevare dati statici necessari all'esecuzione del programma, come i parametri di configurazione. [22]

**ROS1-ROS2 Bridge**

Essendo ROS2 un ambiente relativamente nuovo, si è ancora in una fase di transizione. Per tale ragione è stato programmato un nodo-bridge che permette di migrare i pacchetti sviluppati in ROS1 verso la nuova versione. E' così possibile

risparmiare nella riscrittura del codice e approfittare delle nuove funzionalità introdotte. [22]

**Utilizzo di ROS2 in progetti critici per la sicurezza**

In tempi recenti ROS2 ha attratto molto interesse da parte di società che si occupano dello sviluppo di veicoli a guida autonoma. Se i prototipi dovessero funzionare, il sistema di librerie open source di ROS2 non sarebbe in linea con le norme di sicurezza (ISO 26262) richieste dal mondo automotive, per questo le società di infrastrutture si sono attivate per adattare le librerie all'esecuzione di azioni in real-time. In aggiunta, le società hanno iniziato a produrre librerie di ROS2 che possono essere usate in progetti in cui la sicurezza è criticamente importante. [21]

**ROS1 VS ROS2**

ROS2 Galactic Geochelone è una delle ultime distribuzioni rilasciate, creato per un utilizzo commerciale e progettato con l'intento di semplificare l'hardware acceleration durante il controllo dei robot, ma mantenendo intatte le funzionalità di base di ROS1. [23]

Di seguito sono elencati i miglioramenti apportati:

- Implementazione della sicurezza (assente in ROS1);
- Supporto per sistemi multi-robot e miglioramento della loro intercomunicazione;
- Supporto di driver per il funzionamento "real time"
- Miglioramento delle prestazioni di rete, performance e scalabilità;
- Miglioramento delle prestazioni e della tempestività di controllo;
- Aumento del numero di sistemi operativi supportati.

**Quality of Service**

L'aggiunta del DDS in ROS2 introduce nuove politiche di Servizio garantendo diversi livelli di affidabilità nella comunicazione tra nodi. Combinando le policy ROS2 può risultare affidabile sfruttando il protocollo TCP o fornire un servizio di consegna "best-effort" attraverso il protocollo UDP. [16] [24]

Nella tabella sottostante vengono presentate le policy introdotte in ROS2.

- *Deadline:* Tempo massimo che intercorre tra due messaggi pubblicati o ricevuti

- *Storia:* Definisce la quantità di pacchetti mantenuti in memoria. Possibilità di salvarli tutti (a seconda della quantità di memoria disponibile), oppure solo gli ultimi N pacchetti.

- *Affidabilità:* Nel "Best-Effort" la trasmissione dei dati avviene il più velocemente possibile ma non è garantita la consegna. Una trasmissione Reliable assicura l'arrivo dei pacchetti a destinazione, a costo di ritrasmettere più volte il pacchetto.

- *Durata:* Il servizio tenta di mantenere più messaggi possibili fino al raggiungimento della capienza massima della Storia così da poter essere consegnati verso eventuali subscriber "lenti".

### 2.2.2 MoveIt2

#### Introduzione e caratteristiche

Recentemente si è visto un rapido sviluppo dell'industria robotica e un altrettanto aumento della richiesta di robot. La loro applicazione spazia tra moltissimi ambiti avvicinandosi anche a quelli in cui è richiesta una stretta collaborazione con l'uomo, condividendo mansioni e spazio di lavoro. Ne sono un esempio i robot domestici che devono effettuare movimenti il più possibile precisi e controllati, considerando un ambiente di lavoro variabile e possibilmente affollato, oltre a presentare oggetti fragili o pericolosi. Per questioni di sicurezza e di affidabilità risulta quindi necessario sfruttare un programma di supporto.
MoveIt è un insieme di pacchetti software integrabili in ROS2, creato specificatamente per manipolazione di robot con scopi industriali, commerciali e di ricerca. Incorpora gli ultimi aggiornamenti per il Motion Planning, la manipolazione, la percezione 3D, la movimentazione, il controllo e l'attuazione. Permette quindi al robot in studio di valutare tramite opportuni sensori l'ambiente di lavoro, ed eventualmente di mapparlo; generare una proposta di esecuzione di un determinato movimento; quindi una traiettoria ben definita ed un'esecuzione in ambiente monitorato. [25]
MoveIt può lavorare con oltre 150 robot differenti, numero in costante aumento, per questo è stato creato MoveIt Setup Assistant, un tool grafico per semplificare la fase di setup di una configurazione personalizzata o utilizzarne di preimpostate.
Molte aziende di elevata importanza come Nasa, Microsoft, Google, Samsung,

Pal Robotics, Franka Emika ne fanno utilizzo. [26]
Vengono di seguito elencate le principali funzionalità:

- *Motion planning:* Genera un elevato grado di libertà delle traiettorie attuabili anche attraverso ambienti;
- *Manipulation:* analizza e interagisce con l'ambiente circostante grazie ad una famiglia numerosa di utensili robotici per la manipolazione;
- *Inverse kinematics:* Modalità che permette la computazione e risoluzione di un problema di posizionamento per una data situazione finale fornita;
- *Control:* Permette la parametrizzate nel tempo delle traiettorie;
- *3D Perception:* Permette una visualizzazione 3D attraverso opportuni sensori di profondità;
- *Collision checking:* Evita le collisioni attraverso funzioni preimpostate.

**Pianificazione del movimento**

Il calcolo del movimento tra due pose differenti consiste nella produzione di una sequenza di angoli che ogni giunto del braccio robotico deve seguire, coordinato con gli altri giunti, senza entrare in collisione con eventuali oggetti presenti nell'ambiente di lavoro.
A seguito della richiesta di movimento il nodo Move Group genera in risposta una traiettoria. Delle innumerevoli calcolabili, vengono eliminate automaticamente quelle che presentano collisioni [27]. Eventualmente la richiesta può essere modificabile secondo dei vincoli detti "vincoli cinematici" basati su:

- *Posizione:* limita la posizione di un link all'interno di una regione di spazio;
- *Orientazione:* limita l'orientazione di un link secondo specifici valori di rollio beccheggio o imbardata
- *Visione:* limita un punto di un link entro un determinato cono di visione per un particolare sensore;
- *Giunti:* limita il movimento di un giunto tra due valori specifici;

**Verifica delle collisioni:** in MoveIt viene configurata all'interno della Planning Scene (ambiente simulato) e gestita da un pacchetto di librerie denominato FCL

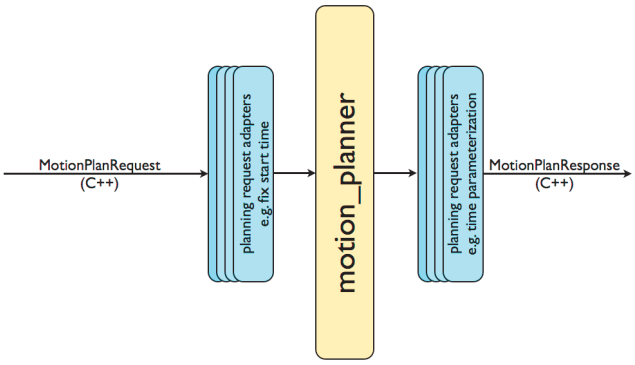


**Figura 2.13:** Motion planner pipeline

(Flexible Collision Library) che si occupa di verificare se sono presenti collisioni ancor prima di formulare la richiesta di pianificazione del movimento.
Le collisioni in moveit possono avvenire tra diverse tipologie di oggetti:

- *Meshes:* corpi complessi generati da unioni di superfici poligonali;
- *Oggetti primitivi:* parallelepipedi, cilindri, coni e sfere;
- *OctoMap:* oggetti generati da un framework di mappatura 3D che riconosce tramite una telecamera l'ambiente circostante e lo ricrea con tridimensionalità all'interno dell'ambiente virtuale.

Se nell'ambiente di lavoro si hanno coppie di oggetti che, per motivi di costruzione o di progetto non possano mai andare in contatto (come ad esempio due bracci robotici posizionati ad una distanza maggiore della somma delle loro massime estensioni), si consente la collisione attivando la ACM (Allowed Collision Matrix) tra di loro. Questa operazione viene eseguita nel caso si voglia alleggerire il carico di lavoro computazionale molto spesso esoso. [28]
**Tipologie di cinematica:** la cinematica di un braccio robotico è riferita allo studio del moto non considerando forze e momenti che lo causano.
Un manipolatore può essere visto come interconnessione di segmenti rigidi attraverso dei giunti, partendo dalla base per raggiungere l'end effector.
Esistono due tipi di cinematica per lo studio di un sistema robotico, quella diretta e quella inversa.
La cinematica diretta è un problema che riguarda la determinazione di posizione e orientamento dell'end-effector avendo note le lunghezze dei link che compongono il manipolatore e gli angoli dei giunti. Viene risolto associando ad ogni membro una terna di riferimento ad esso solidale che ne descrive la configurazione, e utilizzando trasformazioni omogenee per la descrizione della posizione di una terna

rispetto alla precedente. [29] [30]

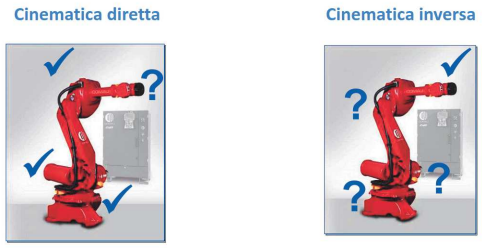


**Figura 2.14:** Differenza tra cinematica inversa, e cinematica diretta [31]

La cinematica inversa invece è un problema di calcolo delle configurazioni dei giunti del sistema robotico necessarie a portare l'end-effector nella posizione e orientazione desiderata.
Non esiste alcuna tecnica di tipo generale da poter applicare sistematicamente ma, solitamente, vengono cercate soluzioni in forma chiusa e non numeriche evitando la computazione di equazioni che risulterebbero assai complesse. Inoltre, a seconda dei casi di studio, è possibile avere un insieme finito o infinito di soluzioni, o non presentare alcuna.Di fatto solo alcuni manipolatori costituiti da catene cinematiche "semplici" hanno una soluzione analitica completa, poiché aumentando il numero di gradi di libertà della catena cinematica, il numero di calcoli necessari cresce esponenzialmente. [32] [33]

### 2.2.3 Libreria Tf2

Quando un sistema robotico effettua un movimento nello spazio ognuno dei suoi giunti modifica la sua posizione rispetto alla base. Tf2 è una libreria utilizzata in ROS2 che tiene conto delle relazioni tra i sistemi di coordinate (*frames*) secondo una gerarchia ad albero e le mantiene aggiornate nel tempo, permettendo di conoscere sempre a quale frame sono riferite le posizioni dei vari elementi dell'ambiente.
Operando in un sistema distribuito è possibile salvare le relazioni tra i frame in un server centrale piuttosto che distribuirle a tutti i nodi del sistema in modo da renderle più facilmente accessibili.

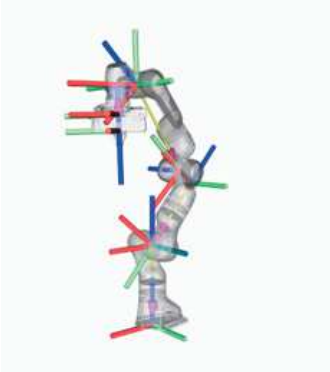

**Figura 2.15:** Visualizzazione dei sistemi di riferimento dei link

La libreria Tf2 è costituita da due moduli, un Broadcaster ed un Listener. Il primo condivide a tutto il sistema informazioni sulle trasformate; Il secondo è in grado di ricevere queste informazioni, salvarle, ed è eventualmente responsabile della risposta alle richieste di trasformata tra due frame [34]. L'ambiente di visualizzazione RViz è uno strumento utile per esaminare i frames della libreria Tf2. Ogni frame infatti ha una sua posizione nello spazio riferita al sistema di riferimento principale *"world"* ma anche una sua posizione relativa rispetto ad un altro. Il visualizzatore permette di avere un'idea più chiara della disposizione degli assi. [35] [36]

### 2.2.4 AprilTag

Gli AprilTag sono un sistema di visione fiduciario sviluppato per la prima volta da Ed Olson, utilizzabile in una vasta gamma di operazioni tra cui realtà aumentata, robotica e calibrazione di fotocamere.
Forniscono un sistema di identificazione e posizionamento 3D anche in situazioni di scarsa visibilità o con difficili angoli di visuale. Il concetto è simile a quello di un QR code o di un codice a barre: racchiudendo solamente l'informazione dell'ID del Tag, permette la stima della sua posizione in coordinate 6D (x, y, z, rollio, beccheggio, imbardata) rispetto alla posizione della fotocamera dalla quale viene decodificato. Solo nel caso in cui quest'ultima venga calibrata e si conosca la dimensione fisica del tag, ne si riesce a ricavare la trasformata.
Esistono in totale 6 famiglie di AprilTag come da figura 2.16. Ognuna di esse è composta da una vasta gamma di immagini differenti secondo un diverso tipo di "mosaico" (figura 2.16).

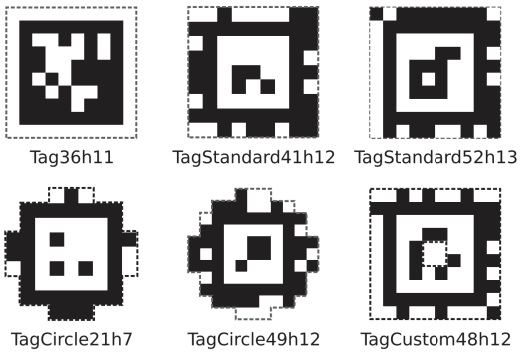


**Figura 2.16:** Famiglie di AprilTag [37]

Tramite diversi esperimenti è stato notato che la precisione di posizionamento è fortemente dipendente qualità della fotocamera e dalla distanza che essa ha rispetto al Tag preso in questione. [38] [39]

# Capitolo 3

# Setup sperimentale

Il capitolo seguente descrive la realizzazione pratica del progetto di Pick and Place di alcuni semplici oggetti cubici mediante robot collaborativo Franka Emika Panda. La prima sezione consiste nella descrizione del setup dei tool software per simulare l'intero progetto all'interno del proprio personal computer, in cui è installato il sistema operativo Ubuntu in quanto già stabilmente supportato e ben fornito. La seconda sezione riguarda invece l'implementazione in ambiente reale del progetto di cui seguiranno prove pratiche a fini di ricerca.

## 3.1 Descrizione generale

Seguendo la precedente descrizione dell'ambiente ROS, per la creazione del programma di controllo è necessario creare un workspace in cui sono contenuti tutti i file.
Un workspace è una cartella con una particolare struttura, all'interno della quale vengono installati tutti i pacchetti che costituiscono la rete del sistema.
Per l'esecuzione del build contemporaneo e automatizzato di più pacchetti in una singola chiamata viene utilizzato il build tool *Colcon* grazie al quale nel workspace vengono generate diverse sottocartelle: **src**, **install**, **build** e **log**. Nella prima vengono installati tutti i codici sorgente; nella seconda sono installati i pacchetti ognuno in una differente sottocartella; negli ultimi due, rispettivamente, vengono salvati i file temporanei associati ad ogni pacchetto e le informazioni riguardo ogni chiamata del comando *colcon build*.
Per questo progetto è stato necessario definire nodi per la gestione del movimen-

to, della fotocamera, delle trasformate tra vari frame e per l'identificazione degli AprilTag con cui ci si interfaccia all'ambiente reale.

## 3.2 Progettazione

La prima richiesta per la progettazione di un sistema Pick & Place è la scrittura del codice di controllo. Si è presentata la scelta tra due diversi linguaggi di programmazione, C++ o Python. Essi hanno caratteristiche molto diverse ma entrambe valide per l'esecuzione del progetto. La scelta è ricaduta verso il linguaggio C++ nonostante la sintassi più complessa, perché risulta più adatto ad applicazioni di manipolazione avanzate grazie ad una maggiore velocità di esecuzione e di maggiore diffusione in ambiti industriali. Differentemente da Python, che è un linguaggio interpretato, quindi senza generazione di un codice eseguibile, C++ è compilato ed esegue il debugging durante la compilazione evitando l'avvento di crash anomali potenzialmente pericolosi in fase di sperimentazione reale.
La presenza di numerose librerie fornite da ROS e dalla sua community semplifica e organizza la scrittura del codice di controllo.
Già presente nel sistema Moveit al momento dell'installazione è la classe MoveGroupInterface, un'intuitiva interfaccia utente che comunica su ROS, con il nodo MoveGroup, attraverso i communication patterns (topic, service, action). Quest'ultimo è il nodo primario di MoveIt che collega tutti i singoli componenti fornendo all'utente un insieme di servizi e azioni per l'intercomunicazione.
MoveGroupInterface dispone anche di funzioni per ricavare informazioni sullo stato del robot, o visualizzare ed impostare alcuni parametri della simulazione, ad esempio velocità e tempo massimo di pianificazione del movimento.
Nella funzione di controllo il sistema robotico è suddiviso in due gruppi di giunti (joint group), chiamati "Panda Arm" e "Hand", rispettivamente corrispondenti ai sette giunti di cui è composto il braccio e al controllo della pinza robotica.
Prima di effettuare un movimento è richiesta la sua pianificazione nello spazio. Esistono due diverse modalità:

- definendo un valore assoluto per l'angolo finale che deve raggiungere ogni giunto della catena cinematica;
- definendo la posizione e orientamento nello spazio che deve raggiungere l'end effector.

Si è deciso di seguire il secondo metodo di pianificazione perché più adatto alle applicazioni di Pick and Place.
L'approccio verso la movimentazione del robot è stato graduale. I primi test sono stati effettuati tentando di portare il manipolatore in posizioni casuali all'interno dello spazio di lavoro per familiarizzare con l'ambiente e la logica di movimento. Una volta che l'attività di posizionamento è stata migliorata, mediante delle funzioni della classe PlanningSceneInterface sono stati generati ed implementati i blocchi virtuali da manipolare. Questi vengono definiti come *Collision Objects*, ovvero oggetti fisici che non possono entrare in collisione con il robot durante l'esecuzione.
Inizialmente l'attività di manipolazione è stata effettuata conoscendo le esatte coordinate dei blocchi e di conseguenza la posizione di arrivo dell'end-effector, tenendo conto della distanza tra la giunzione e i gripper di quest'ultimo.
Dopo numerosi tentativi di pianificazione è stato notato che in alcuni casi la traiettoria compiuta risultava inefficiente e spesso inaspettata. Per migliorare tale situazione e permettere un maggior controllo si è implementata un'ulteriore funzione della classe MoveGroup, che obbliga all'end effector un percorso lineare seguendo una serie di coordinate intermedie definite precedentemente in un vettore.
Anche in questo caso però le traiettorie portavano a movimenti innaturali.
Approfondendo lo studio del Motion Planning si è ipotizzato che l'algoritmo di cinematica inversa per la computazione del movimento non fosse adeguato (*KDL IK solver*), si è quindi deciso di ricercarne uno migliore (*Track_IK solver*).
Nel mentre è stata implementata la libreria TF2 per poter pubblicare una trasformata statica in posizione casuale e calcolarne quella assoluta dopo aver sottoscritto il topic del publisher. Si è deciso di fissare il *base frame* del robot coincidente con il sistema di riferimento dell'ambiente simulato per facilitare la computazione tra i vari sistemi di coordinate. È stato così possibile simulare il picking di blocchi posizionati casualmente all'interno dello spazio di lavoro.
Infine nel passaggio all'ambiente reale si ricorre alla mappatura, dello stesso, mediante calibrazione con una telecamera, sfruttando l'identificazione di alcuni AprilTag. I nodi di AprilTag detection e fotocamera si interfacciano attraverso dei topic comunicando al nodo di controllo le trasformate necessarie al calcolo delle posizioni assolute degli elementi, come da figura 3.1.

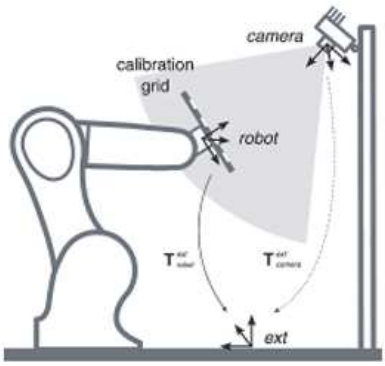


**Figura 3.1:** Hand-Eye Calibration

## 3.3 Ambiente virtuale Rviz

### Descrizione dell'ambiente

Rviz, abbreviazione di ROS Visualization, è il visualizzatore 3D grafico di ROS, uno strumento essenziale per effettuare operazioni di debugging per sistemi robotici.
MoveIt Rviz consente di visualizzare un ambiente di lavoro virtuale, comprensivo del modello 3D del robot o sistema robotico utilizzato. In tale ambiente, utilizzando la classe Planning Scene, sono integrabili figure geometriche primitive o anche ambienti molto complessi con cui il robot può interagire.
Si può quindi visualizzare il sistema nello stato di partenza (*Start State*) e nello stato di arrivo (*Goal State*) rispettivamente di colore verde e arancio di default (come da figura 3.2);

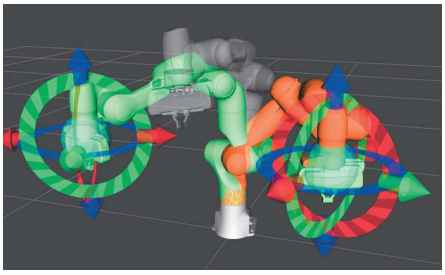

**Figura 3.2:** Start State e goal State

oppure muovere manualmente ogni link costitutivo del sistema e verificare quando si incorre in una collisione tra uno o più componenti (evidenziata in rosso nella figura 3.3) .
Se si effettua una operazione di "Attach Object" per simulare un oggetto fissato ad un determinato end effector l'oggetto diventa di colore viola.

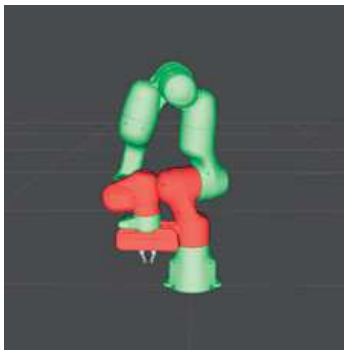

**Figura 3.3:** Stato di collisione

RViz è un'interfaccia grafica intuitiva e presenta all'utente numerosi strumenti grafici di supporto come TF o la visualizzazione dei dati di un determinato sensore o fotocamera implementato; aumentare la velocità di esecuzione, attivare o disattivare la computazione delle collisioni, cambiare sistema di riferimento e molto altro.
È uno strumento essenziale durante la delicata azione del Motion Planning: consente la visualizzazione del movimento pianificato e, una volta valutata la sua fattibilità, ne permette l'esecuzione mostrando, se si desidera, frame per frame la traiettoria (figura 3.4). [40]

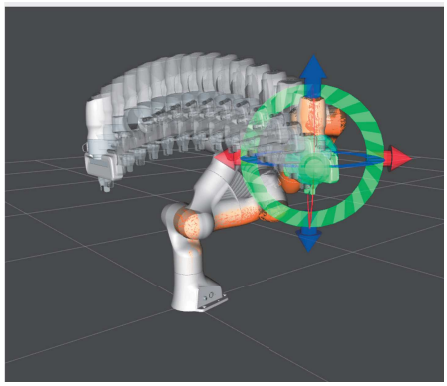

**Figura 3.4:** Visualizzazione della traiettoria in Rviz

### Utilizzo di RViz nel progetto

All'interno dell'ambiente di lavoro simulato di Rviz, per il progetto in tesi, è stato necessario implementare dapprima la visualizzazione del cobot Panda comprensivo della pinza robotica, successivamente due blocchi virtuali di lato 5 cm ad una distanza da esso di circa 50 cm. Tali cubi sono stati posizionati sullo stesso piano

della base del robot, per simulare le condizioni dell'ambiente reale.

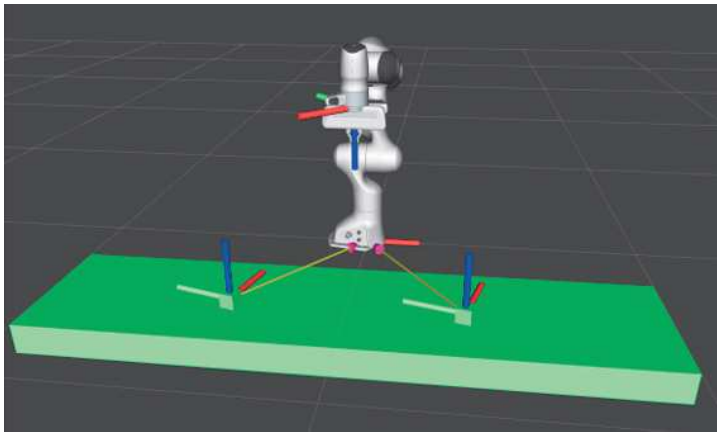

**Figura 3.5:** Rappresentazione dell'ambiente virtuale

Durante l'esecuzione del programma viene richiesto lo spostamento di due blocchi, uno alla volta, in modo che alla fine risultino uno sopra all'altro.
Si è suddiviso il movimento completo del braccio in diversi step, e ogni volta richiesto il clic di un pulsante *next* per procedere con l'esecuzione della traiettoria pianificata, assicurando maggiore sicurezza e controllo scandendo i vari passi e potendo verificare i dati nel caso l'esecuzione non vada a buon fine.

## 3.4 Ambiente reale C-Square Lab

In laboratorio è stato possibile eseguire delle prove di semplici Pick&Place conoscendo la posizione degli oggetti da prelevare, ma senza l'integrazione della telecamera per visualizzare gli AprilTag. Non sono state effettuate ulteriori prove di modifica dell'orientamento dei cubi per motivi di sicurezza.

**Figura 3.6:** Pick and Place con robot reale

# Capitolo 4

# Risultati

## 4.1 Ambiente Virtuale

In questo capitolo vengono descritte le problematiche incontrate durante la realizzazione del progetto e le relative soluzioni adottate.
Nei test iniziali si è notato che posizionando i cubi molto vicino al corpo del Cobot si otteneva una traiettoria in cui esso entrava in conflitto con sé stesso. A beneficio dello studio si è deciso di posizionarli ad una distanza di circa 50 cm, più consona per un movimento reale e meno problematica.
Successivamente è stata riscontrata una discordanza di orientamento degli assi Z tra le TF generate con il publisher e quelle del sistema di riferimento posto sull'end effector, motivo per cui erano necessari dei calcoli aggiuntivi per orientare correttamente la Franka Hand durante il picking (vedere figura 3.5). Per semplificare il problema è possibile calcolare preliminarmente la trasformata della rotazione del frame posto sull'end-effector e mantenerla come parametro fisso durante il calcolo delle traiettorie.
Un altra criticità è stata individuata nella pianificazione del movimento che in alcuni casi portava a traiettorie non consone e pericolose: a volte il braccio ruotava su se stesso, altre si bloccava tentando di attraversarsi, oppure non veniva individuata nessuna traiettoria completa. Studiando in maniera più approfondita l'argomento della cinematica inversa, si è deciso di sostituire l'algoritmo predefinito (KDL) con uno nuovo denominato Track_IK, verificando immediatamente la miglioria apportata.
Nonostante ciò si è notato che in presenza della superficie virtuale rappresentante il tavolo, a volte, anche questo algoritmo non completava il movimento, rimanendo comunque entro i termini di sicurezza. Sono state formulate due ipo-

tesi: la prima secondo cui lo spazio occupato dal tavolo limitava le traiettorie disponibili; la seconda riguardante la collisione tra esso e i cubi da movimentare. Questi ultimi probabilmente nel contatto con la base erano riconosciuti come una compenetrazione tra oggetti con una "fisica di collisione". Dopo alcune prove si è visto che la traiettoria è calcolata fino a quando il blocco attaccato ai gripper si avvicina al tavolo di supporto senza toccarlo. Essendo l'algoritmo in fase di sviluppo è possibile non sia stato ancora integrato l'aspetto del collision checking per gli oggetti afferrati.

## 4.2 Ambiente reale

In ambiente reale sono stati ottenuti i risultati previsti, anche in relazione alla semplicità delle prove che è stato possibile eseguire.
È stato necessario calcolare precisamente lo spessore degli oggetti da afferrare per fare in modo che il sistema considerasse la chiusura della pinza come completa. Anche uno scarto di qualche millimetro poteva portare la pinza a non chiudersi mai interamente e quindi generare un errore terminando l'intera esecuzione del programma.
Definire la manipolazione seguendo traiettorie lineari ha mostrato come il braccio rispettasse un movimento ordinato e sicuro.
Sono state eseguite alcune prove di identificazione degli AprilTag ma senza implementazione nel sistema reale perché non completamente affidabili.

# Capitolo 5

# Conclusione

## 5.1 Virtuale vs Reale

Il progetto, anche se parzialmente riuscito, ha restituito buoni risultati. Si è dimostrato come ROS2 sia un ambiente molto potente e utile in ambito di progettazione e test. In combinazione con MoveIt e RViz risulta uno strumento completo che semplifica enormemente le fatiche del programmatore fornendo un'ampia struttura di base per lo sviluppo di sistemi robotici. Infatti le differenze tra ambiente virtuale e reale sono state minime, dimostrando una grande affidabilità dei software.

## 5.2 Eventuali Sviluppi Futuri

Il problema del Pick&Place risulta molto semplice ma fornisce numerose opportunità di sviluppo; ne vengono successivamente discusse alcune.
Interagendo con una telecamera è possibile realizzare una calibrazione Hand-Eye e generalizzare il problema del picking effettuando il prelevamento di oggetti la cui posizione non è nota, sfruttando l'identificazione attraverso degli Apriltag. Inoltre si ha la possibilità di inserire diversi sensori e telecamere per mappare l'ambiente di lavoro in modo continuo, permettendo ai manipolatori di interagire con un ambiente dinamico.
Una generalizzazione del problema di Pick&Place è la *Teoria del Grasping*, che studia le modalità di presa efficiente di oggetti complessi e privi di simmetrie eseguendo inoltre analisi sull'equilibrio durante lo spostamento.
Spingendosi verso progetti maggiormente articolati si può pensare ad una integra-

zione uomo-macchina più profonda, permettendo collaborazioni come in ambito medico o civile migliorando così le prestazioni dell'operatore e mantenendo un controllo sicuro della macchina. Sfruttando il *machine* learning sarà magari possibile migliorare la pianificazione del movimento del robot rendola ancora più veloce ed affidabile.

# Bibliografia

## Articoli

[3] Ilias El Makrini et al. «Working with walt: How a cobot was developed and inserted on an auto assembly line». In: *IEEE Robotics & Automation Magazine* 25.2 (2018), pp. 51–58 (cit. a p. 6).

[25] Sachin Chitta, Ioan Sucan e Steve Cousins. «Moveit![ros topics]». In: *IEEE Robotics & Automation Magazine* 19.1 (2012), pp. 18–19 (cit. a p. 20).

## Riferimenti bibliografici

[11] Morgan Quigley et al. «ROS: an open-source Robot Operating System». In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5 (cit. a p. 13).

[16] Yuya Maruyama, Shinpei Kato e Takuya Azumi. «Exploring the performance of ROS2». In: *Proceedings of the 13th International Conference on Embedded Software*. 2016, pp. 1–10 (cit. alle pp. 15, 19).

[35] Tully Foote. «tf: The transform library». In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE. 2013, pp. 1–6 (cit. a p. 24).

[39] Edwin Olson. «AprilTag: A robust and flexible visual fiducial system». In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 3400–3407 (cit. a p. 25).

## Libri

[32] Serdar Kucuk e Zafer Bingul. *Robot kinematics: Forward and inverse kinematics*. INTECH Open Access Publisher London, UK, 2006 (cit. a p. 23).

# Sitografia

## Capitolo 2


[1] *Robot collaborativi o cobot: cosa sono?* URL: http://https://www.universal-robots.com/it/robot-collaborativi-o-cobot-cosa-sono-la-guida-definitiva/ (cit. a p. 4).

[2] *Cosa sono i Robot collaborativi e quali sono i vantaggi.* URL: https://www.internet4things.it/iot-library/cobot-cosa-sono-e-quali-sono-i-vantaggi-dei-robot-collaborativi/ (cit. a p. 5).

[4] *Cosa sono e qual è il futuro dei cobot, i robot collaborativi che affiancano gli uomini sulle linee di produzione.* URL: https://www.industry4business.it/industria-4-0/cosa-sono-e-qual-e-il-futuro-dei-cobot-i-robot-collaborativi-che-affiancano-gli-uomini-sulle-linee-di-produzione/ (cit. a p. 6).

[5] *Franka Emika Panda-POMO Robotics.* URL: https://www.pomorobotics.com/robots/frankpanda/ (cit. a p. 8).

[6] *Panda Ecosystem.* URL: https://www.franka.de/ecosystem/ (cit. a p. 10).

[7] *Franka Emika Panda Datasheet.* URL: https://wiredworkers.io/wp-content/uploads/2019/12/Panda_FrankaEmika_ENG.pdf: (cit. a p. 11).

[8] *Franka Production.* URL: https://www.franka.de/production/ (cit. a p. 12).

[9] *Specifiche hardware di Azure Kinect DK.* URL: https://learn.microsoft.com/it-it/azure/kinect-dk/hardware-specification (cit. a p. 12).

[10] *Robot Operating System.* URL: https://it.wikipedia.org/wiki/Robot_Operating_System (cit. a p. 13).

[12] *Appunti di informatica:ROS*. URL: http://www.illuminamente.org/dokuwiki/doku.php?id=neurali:ros (cit. a p. 13).

[13] *All About Circuits: ROS*. URL: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/ (cit. a p. 14).

[14] *What is ROS*. URL: https://ubuntu.com/robotics/what-is-ros (cit. a p. 14).

[15] *ROS 2 Architecture Overview*. URL: https://automaticaddison.com/ros-2-architecture-overview/ (cit. a p. 14).

[17] *Understanding Nodes*. URL: https://docs.ros.org/en/galactic/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (cit. a p. 16).

[18] *Understanding Topics*. URL: https://docs.ros.org/en/galactic/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (cit. a p. 16).

[19] *Understanding Services*. URL: https://docs.ros.org/en/galactic/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html (cit. a p. 17).

[20] *Understanding Actions*. URL: https://docs.ros.org/en/galactic/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html (cit. a p. 18).

[21] *ROS2 Architecture*. URL: https://medium.com/software-architecture-foundations/robot-operating-system-2-ros-2-architecture-731ef1867776 (cit. alle pp. 18, 19).

[22] *Parameter Server*. URL: http://wiki.ros.org/Parameter%20Server (cit. alle pp. 18, 19).

[23] *ROS1 VS ROS2 and pratical applications*. URL: https://www.electronicdesign.com/industrial-automation/article/21215225/electronic-design-ros-vs-ros-2-differences-and-practical-applications (cit. a p. 19).

[24] *ROS2 QoS*. URL: https://docs.ros.org/en/galactic/Concepts/About-Quality-of-Service-Settings.html (cit. a p. 19).

[26] *MoveIt Motion Planning Framework*. URL: https://moveit.ros.org/ (cit. a p. 21).

[27] *Motion Planning Concepts*. URL: https://moveit.picknik.ai/galactic/doc/concepts/motion_planning.html (cit. a p. 21).

[28] *MoveIt Kinematics*. URL: https://moveit.picknik.ai/galactic/doc/concepts/kinematics.html (cit. a p. 22).

[29] *Forward Kinematics – La cinematica diretta*. URL: https://www.meccanismocomplesso.org/forward-kinematics-la-cinematica-diretta/ (cit. a p. 23).

[30] *Cinematica diretta ed inversa di manipolatori seriali - Marco Gabiccini*. URL: http://docenti.ing.unipi.it/gabiccini-m/RAR/04%20-%20Cinematica%20diretta%20ed%20inversa%20manipolatori%20seriali.pdf (cit. a p. 23).

[31] *Cinematica dei Robot - Prof. Paolo Rocco*. URL: https://rocco.faculty.polimi.it/FDR/Cinematica%20del%20robot.pdf (cit. a p. 23).

[33] *Robot Kinematics in a Nuthsell*. URL: https://robocademy.com/2020/04/21/robot-kinematics-in-a-nutshell/ (cit. a p. 23).

[34] *ROSWiki TF2*. URL: http://wiki.ros.org/tf2 (cit. a p. 24).

[36] *TF2 Tutorials*. URL: https://docs.ros.org/en/galactic/Tutorials/Intermediate/Tf2/Tf2-Main.html (cit. a p. 24).

[40] *MoveIt Quickstart in RViz*. URL: https://moveit.picknik.ai/galactic/doc/tutorials/quickstart_in_rviz/quickstart_in_rviz_tutorial.html (cit. a p. 31).